

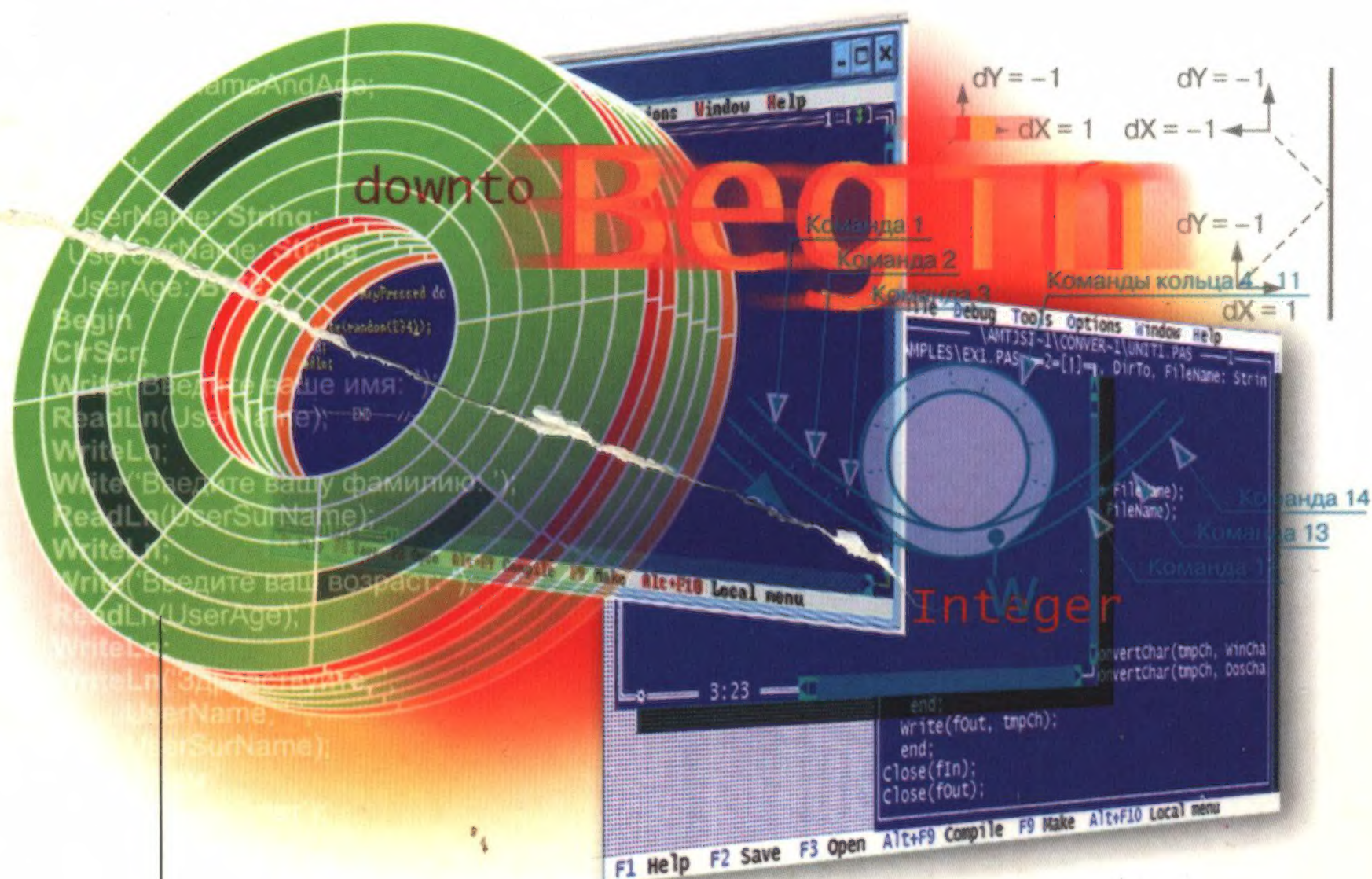
TURBO

Вольский С. В., Дмитриев П. А

PASCAL 7.0

для студентов и школьников

- Основные правила и конструкции языка
- Практика написания программ
Примеры программ с описанием каждой строки кода
- Составление блок-схем
- Шпаргалки (вопросы и ответы) по каждой теме



ПРОСТО О СЛОЖНОМ

Пригодно для подготовки
к зачетам, контрольным и экзаменам.
Лучший выбор для студентов и школьников!

Н и Т
ИЗДАТЕЛЬСТВО

серия — просто о сложном — серия



Наука и Техника

TURBO PASCAL 7.0

**ДЛЯ СТУДЕНТОВ И
ШКОЛЬНИКОВ**



Наука и Техника
Санкт-Петербург
2007

Вольский С.В., Дмитриев П.А.

Turbo Pascal 7.0 для студентов и школьников — СПб.: Наука и Техника, 2007. — 224 с., ил.

ISBN 5-98387-289-2

Серия «Просто о сложном»

Эта книга представляет собой отличное учебное пособие по программированию на языке Turbo Pascal 7.0. Предназначена прежде всего для студентов и школьников, так как содержит весь необходимый объем информации, в то же время не перегружая излишними деталями. Удобная структура и наличие шпаргалок-напоминалок по каждой теме делают книгу незаменимой как при самостоятельном изучении Turbo Pascal, так и при использовании в учебном процессе.

Книга написана доступным языком опытными преподавателями. Содержит большое количество примеров с подробно откомментированными строками программы. Лучший выбор для эффективного изучения Turbo Pascal и подготовки к различным зачетам, контрольным работам и т.п.

ISBN 5-94387-289-2



Контактные телефоны издательства:

(812) 567 70 25, (812) 567 70 26, (044) 516 38 66

Официальный сайт: www.nit.com.ru

© Вольский С.В., Дмитриев П.А.

© Наука и Техника (оригинал-макет), 2007

СОДЕРЖАНИЕ

ГЛАВА 1. ОСНОВНЫЕ ПОНЯТИЯ: ПРОГРАММЫ, АЛГОРИТМЫ, ЯЗЫКИ ПРОГРАММИРОВАНИЯ	11
1.1. ЧТО ТАКОЕ ПРОГРАММЫ. ОТКОМПИЛИРОВАННЫЕ И ИНТЕРПРЕТИРУЕМЫЕ ПРОГРАММЫ	12
1.2. АЛГОРИТМЫ И ЯЗЫКИ ПРОГРАММИРОВАНИЯ	15
1.3. ЯЗЫК ПРОГРАММИРОВАНИЯ TURBO PASCAL.....	17
ГЛАВА 2. ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТЧИКА	19
2.1. ЧТО ТАКОЕ ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТЧИКА	20
2.2. ОСНОВНЫЕ ЭЛЕМЕНТЫ ИНТЕГРИРОВАННОЙ СРЕДЫ РАЗРАБОТЧИКА TURBO PASCAL 7.0	22
Строка оперативной подсказки.....	22
Рабочий стол. Встроенный редактор интегрированной среды	22
Главное меню	23
2.3. САМЫЕ ОСНОВНЫЕ ДЕЙСТВИЯ И КОМАНДЫ В ПРОЦЕССЕ НАПИСАНИЯ ПРОГРАММЫ. БАЗОВЫЕ НАВЫКИ РАБОТЫ С ИНТЕГРИРОВАННОЙ СРЕДОЙ РАЗРАБОТЧИКА TURBO PASCAL 7.0.	24
2.4. СОЗДАНИЕ ПРОГРАММЫ НА TURBO PASCAL 7.0	28
Общая методика	28
Создание файла исходного текста программы.....	29
Сохранение файла с текстом программы.....	29
Создание текста программы. Работа с текстовым редактором среды Turbo Pascal	32
Проверка и отладка текста программы.....	34
Просмотр результатов выполнения программы	36
Создание исполняемого файла программы	36
2.5. ПРАКТИЧЕСКИЙ ПОШАГОВЫЙ ПРИМЕР СОЗДАНИЯ ПРОГРАММЫ	39

ГЛАВА 3. ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ TURBO PASCAL 7.0	43
3.1. ОСНОВНЫЕ ЭЛЕМЕНТЫ ПРОГРАММЫ И АЛФАВИТ ЯЗЫКА	44
3.2 СТРУКТУРА ПРОГРАММЫ	46
3.3. ПЕРЕМЕННЫЕ	48
Что такое переменная. Правила именования переменных в Turbo Pascal	49
Типы данных	50
Оператор присваивания. Совместимость типов данных. Преобразование типов	53
Арифметические и логические выражения.....	56
Механизм работы оператора присваивания.....	58
3.4. КОНСТАНТЫ.....	60
3.5. КОММЕНТАРИИ	61
3.6. СТАНДАРТНЫЕ МАТЕМАТИЧЕСКИЕ ФУНКЦИИ TURBO PASCAL	62
3.7. АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ DIV И MOD	63
3.8. ПОРЯДОК ВЫПОЛНЕНИЯ ОПЕРАЦИЙ В ПРОГРАММАХ НА TURBO PASCAL.....	64
ГЛАВА 4. ВЫВОД ИНФОРМАЦИИ НА ЭКРАН И ВВОД ДАННЫХ В ПРОГРАММУ.....	67
4.1. ВЫВОД ИНФОРМАЦИИ	68
4.2. УПРАВЛЕНИЕ РЕЖИМАМИ ВЫВОДА ИНФОРМАЦИИ.....	70
Процедура ClrScr	71
Функция ReadKey.....	71
Процедуры работы с цветом	72
Процедуры управления расположением информации	75
4.3. ЧТЕНИЕ ИНФОРМАЦИИ С КЛАВИАТУРЫ.....	79
Ввод последовательности символов.....	79
Ввод одиночного символа.....	83

ГЛАВА 5. ОПЕРАТОРЫ ЯЗЫКА	87
5.1. УСЛОВНЫЙ ОПЕРАТОР	88
5.2. СОСТАВНОЙ ОПЕРАТОР.....	90
5.3. ОПЕРАТОР ВЫБОРА.....	91
5.4. ОПЕРАТОРЫ ЦИКЛА	93
Цикл for	93
Цикл while	95
Цикл repeat..until	95
5.5. ОПЕРАТОР БЕЗУСЛОВНОГО ПЕРЕХОДА. МЕТКИ	96
ГЛАВА 6. СЛОЖНЫЕ ТИПЫ ДАННЫХ	99
6.1. МАССИВЫ	100
6.2. СТРОКИ	104
Описание и использование строк.....	104
Процедуры и функции, предусмотренные в Turbo Pascal для работы со строками	105
Сравнение строк.....	108
6.3. ЗАПИСИ	109
Описание записей	109
Операции над записями и полями записей	111
ГЛАВА 7. ПРОЦЕДУРЫ И ФУНКЦИИ	113
7.1. ПРОЦЕДУРЫ.....	115
7.2. ФУНКЦИИ.....	116
7.3. ПРОЦЕДУРЫ И ФУНКЦИИ С ПАРАМЕТРАМИ.....	117
7.4. ПЕРЕДАЧА ПАРАМЕТРОВ ПО ЗНАЧЕНИЮ И ПО ССЫЛКЕ. ПРОЦЕДУРЫ И ФУНКЦИИ С ПАРАМЕТРАМИ-ПЕРЕМЕННЫМИ.....	118
7.5. ПРИМЕР ИСПОЛЬЗОВАНИЯ	119

ГЛАВА 8. РАБОТА С ФАЙЛАМИ.....	123
8.1. ЧТО ТАКОЕ ФАЙЛ. ИМЯ ФАЙЛА	124
8.2. КЛАССИФИКАЦИЯ ФАЙЛОВ В TURBO PASCAL	127
8.3. РАБОТА С ФАЙЛАМИ В TURBO PASCAL	128
Общая методика работы с файлами.....	128
Описание файловой переменной	128
Сопоставление файловой переменной файлу на диске	129
Открытие файла для чтения или для записи	129
Чтение данных из файла и запись данных в файл. Текстовые файлы	130
Закрытие файла.....	132
8.4. ОСОБЕННОСТИ РАБОТЫ С ФАЙЛАМИ СПЕЦИАЛЬНОГО ВИДА	133
Двоичные типизированные файлы	133
Двоичные нетипизированные файлы	134
8.5. ВСПОМОГАТЕЛЬНЫЕ ОПЕРАЦИИ НАД ФАЙЛАМИ.....	135
Определение размера файла.....	135
Изменение текущего элемента	138
8.6. ДОБАВЛЕНИЕ ИНФОРМАЦИИ В ФАЙЛЫ	139
Добавление информации в текстовые файлы	139
Добавление информации в двоичные файлы	140
8.7 ОПЕРАЦИИ С ФАЙЛАМИ И КАТАЛОГАМИ	141
ГЛАВА 9. МОДУЛИ.....	145
9.1. ЧТО ТАКОЕ МОДУЛЬ (БИБЛИОТЕКА) ПОДПРОГРАММ	146
9.2. СТРУКТУРА МОДУЛЯ В TURBO PASCAL 7.0.....	147
ВМЕСТО ЗАКЛЮЧЕНИЯ. ГРАФИКА И ГРАФИЧЕСКИЙ РЕЖИМ	149
ПРИЛОЖЕНИЯ.....	151
ПРИЛОЖЕНИЕ 1. ПОДПРОГРАММЫ МОДУЛЯ CRT.....	152

ПРИЛОЖЕНИЕ 2. ПОДПРОГРАММЫ МОДУЛЯ GRAPH.....	158
ПРИЛОЖЕНИЕ 3. ПОДПРОГРАММЫ МОДУЛЯ SYSTEM	178
Подпрограммы работы с порядковыми типами данных	179
Подпрограммы преобразования типов	181
Арифметические подпрограммы.....	182
Подпрограммы обработки строк	184
Подпрограммы контроля выполнения программы.....	186
Подпрограммы ввода / вывода	187
Подпрограммы работы с текстовыми файлами	190
Подпрограммы работы с нетипизированными файлами	193
Подпрограммы работы с динамической памятью.....	193
Подпрограммы работы с указателями и адресами	195
Другие подпрограммы	196
СПИСОК И ОПИСАНИЕ СООБЩЕНИЙ ОБ ОШИБКАХ	201
Сообщения об ошибках в тексте программы	201
Ошибки времени выполнения.....	211

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100



ГЛАВА 1.

ОСНОВНЫЕ ПОНЯТИЯ: ПРОГРАМ- МЫ, АЛГОРИТМЫ, ЯЗЫКИ ПРО- ГРАММИРОВАНИЯ



1.1. ЧТО ТАКОЕ ПРОГРАММЫ. ОТКОМПИЛИРОВАННЫЕ И ИНТЕРПРЕТИРУЕМЫЕ ПРОГРАММЫ

По своей сути компьютер — лишь электрический прибор, способный только выполнять команды, записанные в виде инструкций. Именно их и называют программами, и ПК без них подобен машине без водителя. Текстовый редактор Word, операционная система Windows, игры и т.д. — все это программы.

Создаются программы на специальных языках, понятных компьютеру (языках программирования), целыми командами профессионалов-программистов, насчитывающими десятки, а то и сотни участников. Обычным же пользователям достаточно знать, как запускать программу и как с ней работать.

При работе с программой пользователь управляет ее работой, а она управляет компьютером. Встречаются программы, которые не предусматривают участие пользователя в их работе, а просто после запуска выполняют заранее predetermined перечень операций и завершают свою работу. Но такие программы встречаются достаточно редко.

В широком смысле компьютерные программы являются средствами автоматизации и моделирования различных процессов. В более конкретном понимании программа — это логически закон-

ченный непрерывный поток команд, выполняемых операционной системой от начала до конца.

Операционная система – это самая главная программа на компьютере (точнее совокупность программ), управляющая всеми остальными программами, которые без нее не смогут функционировать, и обеспечивающая им возможность взаимодействия с устройствами ПК. Кроме того, операционная система является посредником между вами и программами на компьютере: через нее осуществляется запуск всех программ и даются им команды.

Основная часть операционной системы — ядро, определяющее общую функциональность системы и распределяющее процессорное время между выполняемыми программами (в том числе и являющимися частями самой системы). Помимо ядра, в состав операционной системы входит набор драйверов — программ, обеспечивающих доступ операционной системы к внешним устройствам

О том, что без операционной системы ПК не сможет нормально работать, знает и он сам. Поэтому в числе первых действий, которые он осуществляет после включения, — запуск операционной системы, производящийся автоматически.

Самой популярной на сегодняшний день среди российских пользователей является операционная система Windows XP, разработанная американской компанией Microsoft, основатель которой Билл Гейтс — самый богатый человек в мире, сделавший себе имя и капитал на своем детище.

В зависимости от механизма выполнения программы подразделяются на откомпилированные и интерпретируемые. Соответственно языки программирования подразделяются на интерпретируемые и компилируемые:

- **Откомпилированные программы** получают в результате компиляции исходного текста программы и выполняются самой операционной системой. Компиляция заключается в создании из исходного текста — исполняемой програм-

мы и осуществляется специализированной программой — **компилятором** (compiler — компилятор), настроенной на соответствующий язык программирования, или несколько языков программирования сразу. Если программа состоит из нескольких исходных текстов, то в процессе компиляции они все упорядочиваются и объединяются.

- **Интерпретируемые программы** для своего исполнения требуют установку в операционной системе специальной программы-интерпретатора. Операционная система сама по себе не может понять, а значит, и выполнить интерпретируемую программу. В этом ей как раз и помогает интерпретатор, занимающий промежуточное положение между программой и операционной системой и служащий как бы переводчиком.

Особенностью откомпилированной программы является то, что такая программа может выполняться только в операционной системе, под которую была выполнена компиляция. Запуск в другой системе возможен только лишь в том случае, если эта система поддерживает ту, под которую программа была откомпилирована. Таким образом, под разные операционные системы требуется наличие своего персонального компилятора для языка программирования, на котором написана программа.

Если откомпилированная исполняемая программа может работать только в одной операционной системе, то интерпретируемые работают во всех операционных системах, для которых реализован соответствующий интерпретатор. То есть одна и та же программа может быть запущена и работать в любой операционной системе. Нужно лишь, чтобы в этой операционной системе присутствовал интерпретатор, который либо может быть установлен изначально, либо уже по ходу дела вручную. Наглядным примером такого подхода является язык программирования Java, интерпретаторы которого предусмотрены практически для всех операционных систем и стоят даже в большинстве современных сотовых телефонов.

В принципе компилятор и интерпретатор можно сравнить с двумя категориями переводчиков. Компиляторы – это обычные переводчики, работающие по принципу «сдал тест – через некоторое время получи программу». Интерпретаторы же представляют собой своего рода синхронных переводчиков, которые по ходу дела (по ходу выполнения программы) переводят программу строчку за строчкой по мере ее выполнения. Отсюда еще одно преимущество интерпретируемых языков программирования, которое заключается в принципиальной возможности модификации программы уже во время ее выполнения.

Недостатком интерпретируемых программ является их относительная медлительность по сравнению с откомпилированными, а также необходимость в наличии (установке) дополнительного программного обеспечения (интерпретатора), которое забирает для своей работы часть вычислительных ресурсов компьютера.

ПРИМЕЧАНИЕ

Разновидность специальных программ (будь то компиляторы или интерпретаторы), осуществляющих преобразование текста программы в язык машинных кодов, называется **трансляторами**

1.2. АЛГОРИТМЫ И ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Большинство действий, осуществляемых в жизни можно разбить на этапы. Так, например, процедура приготовления овощного салата сводится к следующей последовательности:

1. Сходить в магазин и купить овощей для салата.
2. Помыть овощи.
3. Нашинковать овощи.
4. Заправить маслом или сметаной.
5. Посолить, поперчить.

Совокупность подобных рекомендаций (их называют инструкциями) образует алгоритм решения поставленной задачи. В данном примере задачей служит «Приготовление салата». Алгоритмы служат для передачи их предполагаемому исполнителю, чтобы он, согласно алгоритму, мог выполнить указанную задачу.

Чтобы алгоритм был успешно выполнен, каждый из составляющих его этапов должен быть понятен исполнителю. При этом:

- Инструкции алгоритма должны быть написаны на понятном исполнителю языке.
- Смысл каждой инструкции должен быть однозначно понятен исполнителю.
- Алгоритм должен в принципе обеспечивать результативность, то есть решение поставленной задачи за конечное число этапов.

Например, инструкции типа «1) Пойди туда, не знаю куда. 2) Найди то, не знаю что» хоть и написаны на понятном для русскоговорящего человека языке, но не отвечают требованию однозначности и могут быть либо истолкованы произвольно, либо вообще никак не истолкованы, так как они не обеспечивают требуемой результативности ввиду неясности самой задачи.

Таким образом, **алгоритм – это последовательность четких, понятных и однозначных указаний, разработанных для решения поставленной задачи.**

В случае с компьютером алгоритм, чтобы быть понятным, пишется на одном из языков программирования. При этом то, как он будет написан, зависит уже языка программирования. Разные языки предоставляют разные возможности по написанию программ. Например, если в каком-то человеческом языке нет слова «помыть», то вместо него придется писать множество других слов, поясняющих данное понятие (типа того, что «поливать холодной водой с целью удаления частиц грязи с поверхности»). То же самое и в языках программирования.

Более развитые языки программирования содержат большее количество понятий и языковых конструкций, позволяя обходиться наименьшим количеством «слов» в инструкциях и наибольшим количеством подлежащих описанию действий. Однако, как правило, более развитые языки программирования требуют более высоких временных затрат и умственных усилий для их изучения и понимания. Но впоследствии работать с ними гораздо легче и удобнее.

Принципиально по уровню предоставляемых возможностей (функций) выделяют следующие два класса языков программирования:

- Языки программирования **низкого уровня**, обеспечивающие поддержку простейших операций, обычно на уровне системных устройств — процессора, памяти и т.д. Примером такого языка является язык Ассемблера (Assembler), содержащий только команды пересылки единиц информации между регистрами процессора и областями памяти, а также вызова системных функций операционной системы.
- Языки программирования **высокого уровня**, обеспечивающие поддержку сложных операций, например таких, как получение информации из базы данных по запросу или вывод информации на принтер. В результате использования данных языков получаются более объемные и менее быстродействующие исполняемые программы, что, однако, компенсируется скоростью их разработки и меньшим содержанием ошибок, хотя бы вследствие малых объемов исходных текстов. К таким языкам относятся C/C++, Turbo Pascal, и множество других. В язык высокого уровня может быть встроена поддержка языка низкого уровня, что компенсирует отсутствие в языках высокого уровня доступа к низкоуровневым операциям.

1.3. ЯЗЫК ПРОГРАММИРОВАНИЯ TURBO PASCAL

Язык программирования Turbo Pascal был создан признанным классиком программирования Никлаусом Виртом в начале 70-х

годов прошлого века. Назван язык был в честь великого французского ученого Блеза Паскаля (1623 - 1662), чьи научные открытия охватывают широчайший круг областей человеческого познания и лежат в основе многих современных теорий и представлений в самых различных направлениях науки.

В дальнейшем язык Turbo Pascal развивался американской фирмой Borland, и в том виде, в котором мы пользуемся этим языком сейчас, во многом он обязан этой фирме.

Что касается характеристик языка, то следует отметить прежде всего его понятность и простоту, в то же время сочетающиеся с достаточно развитыми возможностями. Благодаря этим своим качествам Turbo Pascal рекомендован для начального обучения программирования и используется во многих странах мира – в том числе в России и Америке.

Язык Turbo Pascal является языком высокого уровня и широкого назначения. Получаемые в результате разработки исполняемые программы являются откомпилированными и предназначены для выполнения в операционной системе MS-DOS. Однако, ввиду поддержки MS-DOS операционной системой Windows, и из под Windows созданные на Паскале программы вполне успешно запускаются и работают.

ПРИМЕЧАНИЕ

При этом в Windows сначала инициализируется сеанс работы MS-DOS, в котором уже запускается созданная на Turbo Pascal программа.

ГЛАВА 2.

ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТЧИКА



2.1. ЧТО ТАКОЕ ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТЧИКА

Для написания исходных текстов программ практически все современные языки программирования снабжаются так называемыми интегрированными средами разработчика (Integrated Developer's Environment — **IDE**). Эта среда представляет собой своего рода текстовый редактор, обладающий рядом дополнительных возможностей по обеспечению удобства программирования и максимальной автоматизации этого процесса. Кроме того, среда разработчика включает в себя **встроенный компилятор**, позволяющий откомпилировать исходные тексты программы и получить готовый исполняемый файл, а также **встроенный отладчик**, предназначенный для отладки программы.

Чтобы запустить интегрированную среду разработчика Turbo Pascal 7.0, следует запустить одну из следующих программ, входящих в стандартную поставку: `turbo.exe` или `bp.exe`. Интегрированная среда у них обоих выглядит одинаково, обе они выполняют одни и те же функции, отличаясь лишь небольшими деталями в интегрированной среде. При этом среда, запускаемая программой `bp.exe`, является более новой по сравнению с интегрированной средой, запускаемой файлом `turbo.exe`, и предоставляет более развитые возможности. Однако эти возможности касаются лишь процедуры компиляции программ (например, компиляции программ, исполняемых в защищенном режиме про-

цессора) поэтому начинающим программистам рекомендуется запускать интегрированную среду разработчика с помощью файла `turbo.exe`.

Внешний вид окна интегрированной среды сразу после запуска показан на рис. 2.1. Вверху располагается главное меню, основную часть занимает рабочий стол, а внизу располагается строка оперативной подсказки. Подробно эти элементы среды разработчика языка Turbo Pascal будут рассмотрены далее, в следующем параграфе.

Скорее всего запускаться среда разработчика Turbo Pascal 7.0 будет под Windows, а поскольку среда работает под MS-DOS, то сначала, как уже говорилось, будет запущен сеанс работы операционной системы MS-DOS, а уже в его окне открыта среда Turbo Pascal 7.0. При этом, чтобы среда открывалась во весь экран, следует щелкнуть правой кнопкой мыши по заголовку окна и в появившемся контекстном меню выбрать пункт **Свойства**. В результате на экране появится одноименное диалоговое окно, в котором нужно будет установить переключатель **Во весь экран**.

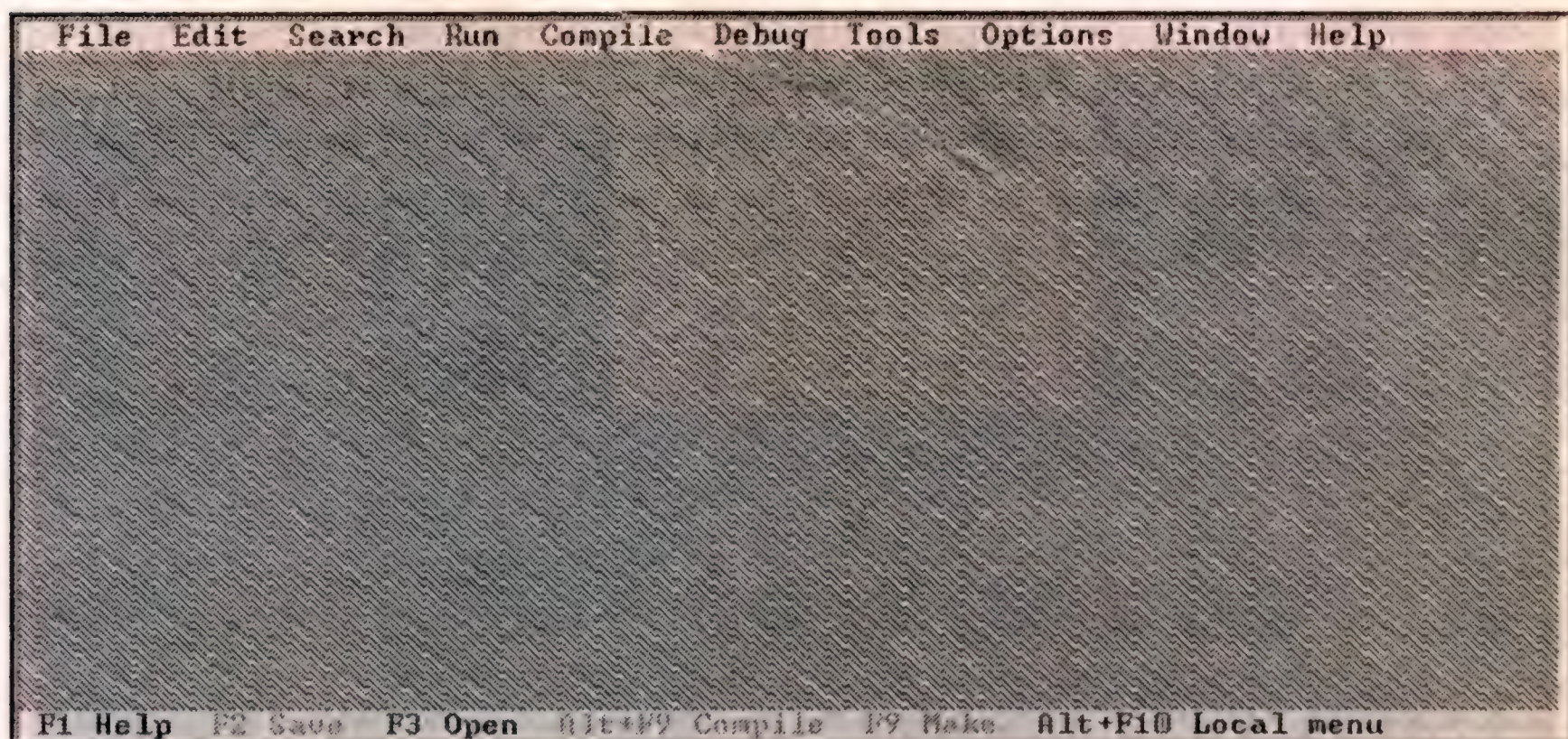


Рис. 2.1. Интегрированная среда разработчика Turbo Pascal после запуска

2.2. ОСНОВНЫЕ ЭЛЕМЕНТЫ ИНТЕГРИРОВАННОЙ СРЕДЫ РАЗРАБОТЧИКА TURBO PASCAL 7.0

СТРОКА ОПЕРАТИВНОЙ ПОДСКАЗКИ

О строке оперативной подсказки сказать особо нечего. Служит она для отображения краткого описания действий, которые вы можете предпринять в данный момент. Кроме того, в строке оперативной подсказки выводится краткое описание выбранного в данный момент пункта меню.

РАБОЧИЙ СТОЛ. ВСТРОЕННЫЙ РЕДАКТОР ИНТЕГРИРОВАННОЙ СРЕДЫ

Рабочий стол представляет собой пространство, на котором может располагаться одно или несколько «окон» с исходными текстами разрабатываемой программы (или программ). В каждом окне редактируется текст одного файла.

На рис. 2.2. показан стандартный вид интегрированной среды разработчика с открытыми в ней для редактирования текстами программам.

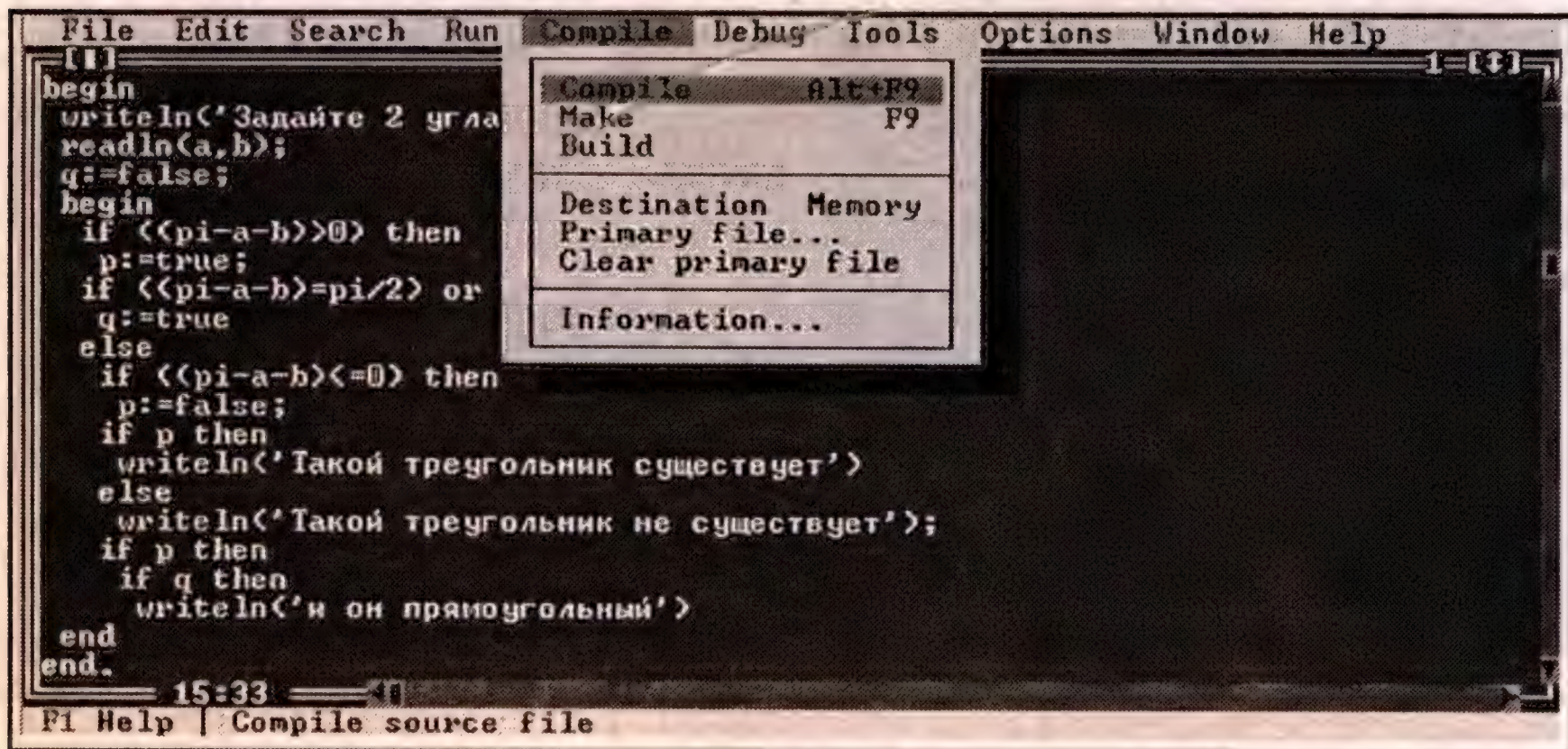


Рис. 2.2. Интегрированная среда разработчика Turbo Pascal в ходе работы

ПРИМЕЧАНИЕ

В принципе в нескольких окнах можно открыть один и тот же файл и производить его редактирование. Однако делать этого не рекомендуется, так как могут возникнуть ошибки, вызванные путаницей в том, в каком именно из окон производились какие изменения. Тем не менее открытие одного файла в нескольких окнах может быть полезным, например, когда нужно иметь перед глазами несколько частей одной и той же программы, текст которой очень большой.

Окна с редактируемыми файлами могут иметь произвольные размеры и положение, а также частично или полностью перекрывать друг друга. Работа с окнами производится точно так же, как и в любой многооконной системе. Например, с помощью мыши можно передвигать окна, менять их размер и т.п.

ГЛАВНОЕ МЕНЮ

Расположено вверху интегрированной среды главное меню содержит следующие пункты:

- **File** — в этом пункте сосредоточены функции по управлению файлами с текстами программ: открытие, сохранение на диске, печать текста программы — все это производится с помощью функций, содержащихся в данном пункте.
- **Edit** — данный пункт содержит функции работы с буфером обмена интегрированной среды, а также функции отката (отмены) нежелательных изменений.
- **Search** — с помощью функций, имеющихся в данном разделе главного меню, можно быстро и удобно осуществлять поиск нужного места в программе (например, поиск нужной функции и т.п.)
- **Run** — здесь сосредоточены функции выполнения и отладки программ.
- **Compile** — данный пункт главного меню образуют функции компиляции — процесса получения исполняемых файлов (*.exe) из исходных текстов программ.

- **Debug** – здесь функции, облегчающие и оптимизирующие процесс отладки программ.
- **Tools** — вспомогательные инструменты, выполняющие некоторые функции, не включенные в интегрированную среду, либо обладающие более широкими возможностями по сравнению с ней, такие как Turbo Assembler, Turbo Debugger и другие.
- **Options** — функции настройки интегрированной среды, такие как настройки компиляции, линковки, отладки, а также визуального представления интегрированной среды на экране.
- **Window** — функции, необходимые для организации окон текстового редактора на экране для более эффективной работы с исходными текстами.

Роль главного меню в процессе разработки программы очень велика, так как именно через него осуществляются основные операции над «окнами» открытых файлов с текстами программ, а также операции по компиляции и отладке.

2.3. САМЫЕ ОСНОВНЫЕ ДЕЙСТВИЯ И КОМАНДЫ В ПРОЦЕССЕ НАПИСАНИЯ ПРОГРАММЫ. БАЗОВЫЕ НАВЫКИ РАБОТЫ С ИНТЕГРИРОВАННОЙ СРЕДОЙ РАЗРАБОТЧИКА TURBO PASCAL 7.0.

Вообще все действия в интегрированной среде можно выполнить с помощью главного меню и клавиатуры. Далее приведены наиболее основные и нужные из них:

- Создавать новые тексты программ – команда **File → New**, открывать уже имеющийся файл с текстом программы для редактирования – команда **File → Open**, сохранять тексты программ после создания и/или редактирования – команда **File → Save**.
- Переключаться между окнами с исходными текстами программ можно с помощью выбора **Window → Next** или

Window → Previous. Соответственно осуществляется переход либо к следующему, либо к предыдущему открытому окну с текстом программы.

- Изменять положение и размер окон с исходными текстами программ с помощью выбора **Window → Size/Move**, или **Window → Zoom**).
- Отменять последние действия, если они были признаны ошибочными с помощью команды **Edit → Undo**. При этом если в какой-то момент сама отмена была произведена по ошибке, то можно вернуться к исходному состоянию (предшествовавшему отмене) выбрав **Edit → Redo**.
- Отправлять на печать текст программы с помощью команды **File → Print**. При этом с помощью команды **File → Printer setup** можно сначала настроить параметры печати, вызвав для этого специальное диалоговое окно.
- Копировать и перемещать фрагменты текста между различными окнами разных программ, тем самым копируя или перемещая фрагменты одной программы в другую. При этом копирование/перемещение производится с помощью внутреннего буфера – специальной области памяти, в которую помещаются данные в промежутке между командой копирования/перемещения и командой вставки. В этом смысле копирование/перемещение фрагментов программы ничем не отличается от копирования/перемещения фрагментов текста в каком-либо текстовом редакторе, например, в том же Word. Сначала копируемый/перемещаемый фрагмент программы выделяется, а затем с помощью команд **Edit → Copy** или **Edit → Cut** указывается, что с ним нужно делать: копировать или вырезать соответственно. Далее курсор помещается в то место, в которое должна осуществляться вставка выделенного фрагмента, и в главном меню выбирается команда **Edit → Paste**. В результате в место нахождения курсора будет перемещен выделенный ранее фрагмент программы или

вставлена его копия в зависимости от того, какая команда – **Cut(Вырезать)** или **Copy (Копировать)** – была выбрана в начале.

- Искать нужный фрагмент в тесте программы и удобно осуществлять навигацию по ней. Например, с помощью команды **Search → Find** можно задать фрагмент текста (строку), которую нужно найти, и она будет найдена (если она вообще в программе присутствует). При этом появляется диалоговое окно **Find**, показанное на рис. 2.3. Если же требуется не просто найти какую-либо строку (слово) в программе, но и заменить ее на другую, то для этого следует воспользоваться командой **Search → Replace**. При этом на экране появится диалоговое окно **Replace** (см. рис. 2.4.), в поле **Text to find** указывается искомая и заменяемая строка, а в поле **Prompt to replace** – строка, на которую должна быть произведена замена. Наконец можно просто перейти к требуемой определенной строке программы, выбрав команду **Search → Go to line number** и в появившемся диалоговом окне введя номер строки, в которую следует произвести переход.
- Запускать программы на выполнение с помощью команды **Run → Run**. По данной команде компилируется и запускается на выполнение программа из текущего активного окна.

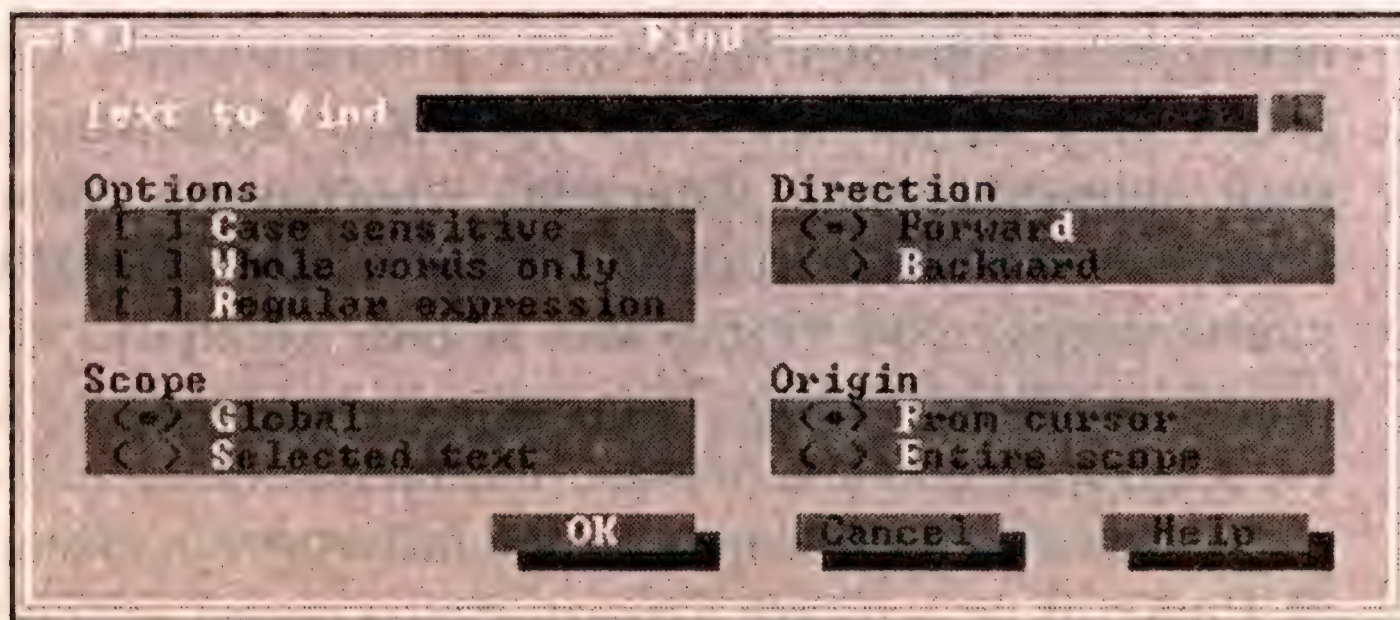


Рис. 2.3. Диалоговое окно *Find*

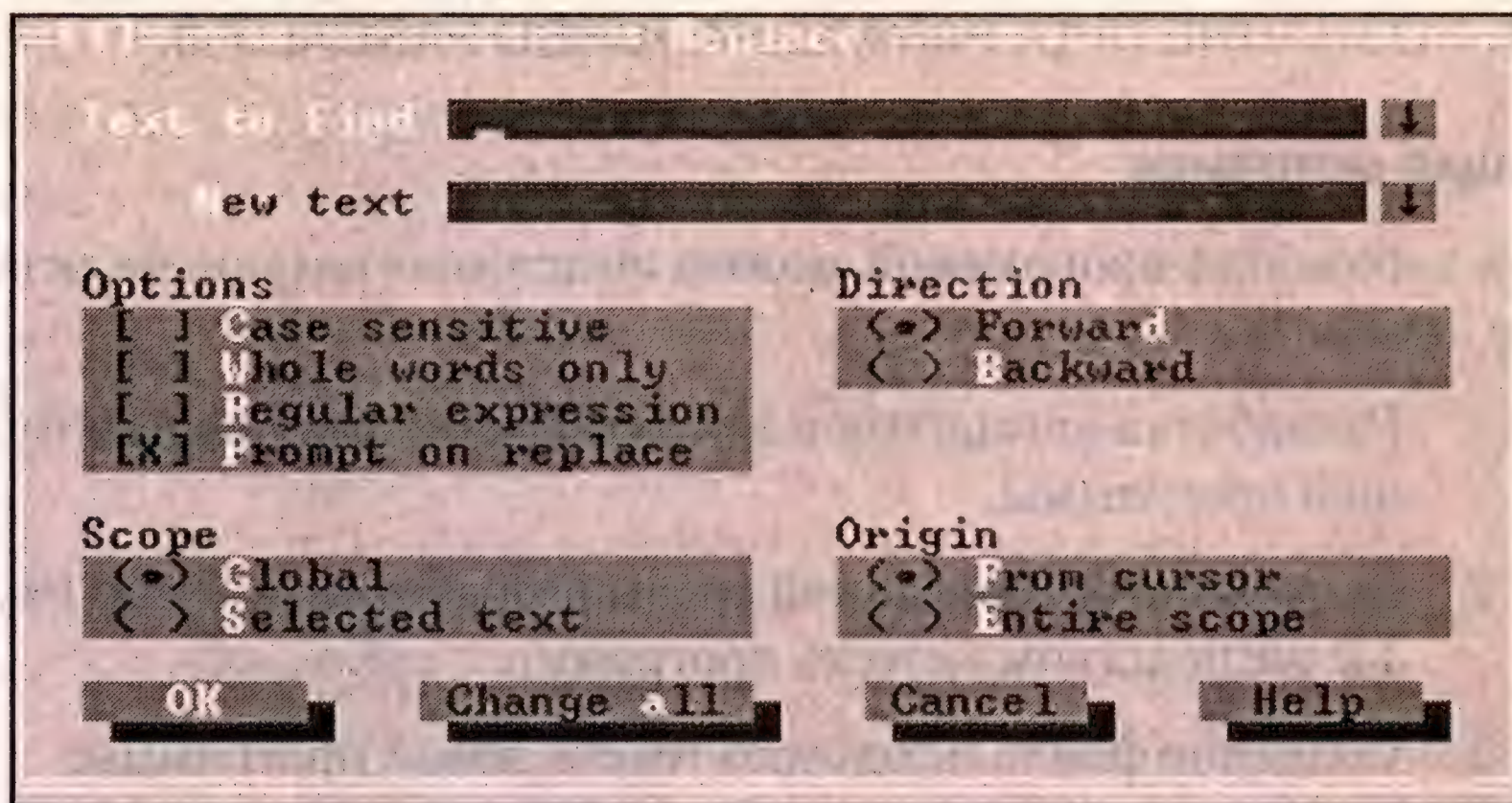


Рис. 2.4. Диалоговое окно Replace

- Компилировать текст программы (без ее запуска) в целях проверки наличия синтаксических ошибок и/или получения уже готового исполняемого файла программы. Произвести компиляцию без получения конечного исполняемого файла можно с помощью команды **Compile → Compile**, с получением исполняемого файла – с помощью команды **Compile → Make**.
- Настраивать параметры интегрированной среды (цвета оформления, часто используемые операции и т.п.) с помощью команд **Options → Environment → Editor** и **Options → Environment → Colors**.

Таким образом, несмотря на то, что интерфейс интегрированной среды разрабатывался достаточно давно (более десятилетия назад), она имеет все необходимые для работы возможности, свойственные современным аналогичным системам.

2.4. СОЗДАНИЕ ПРОГРАММЫ НА TURBO PASCAL 7.0

ОБЩАЯ МЕТОДИКА

При работе над программой можно выделить следующие основные этапы:

1. Разработка алгоритма и продумывание реализации будущей программы.
2. Запуск интегрированной среды разработчика Turbo Pascal 7.0 посредством запуска программы `turbo.exe`.
3. Создание файла исходного текста новой программы.
4. Набор текста программы в окне, соответствующем новой программе.
5. Проверка текста программы на наличие синтаксических ошибок.
6. Проверка программы на предмет правильности ее работы и выполнения задачи, ради которой писалась программа.
7. В случае необходимости создание исполняемого файла программы (этот файл, а значит и сама программа, может быть запущен самостоятельно, без использования интегрированной среды разработчика. Файл же исходного текста может быть запущен только из интегрированной среды (с промежуточной компиляцией естественно)).
8. Сохранение файла с исходным текстом программы. Это сохранение рекомендуется проводить и на промежуточных этапах создания программы, чтобы в случае какого-либо сбоя в работе компьютера не были утеряны результаты уже проведенной работы.

Далее в этом параграфе мы остановимся подробнее на каждом и приведенных этапов за исключением второго, так как о запуске интегрированной среды разработчика уже было сказано ранее.

СОЗДАНИЕ ФАЙЛА ИСХОДНОГО ТЕКСТА ПРОГРАММЫ

Создание нового файла, в котором будет производиться набор, редактирование и сохранение исходного текста программы, производится выбором команды **File → New** из главного меню. В результате будет создан пустой файл с именем NONAME00.PAS.

Если вы в дальнейшем будете создавать еще файлы, то они будут получать имена NONAME01.PAS, NONAME02.PAS и т.д. Обратите внимание, что все они находятся лишь в оперативной памяти компьютера и, чтобы ими можно было воспользоваться впоследствии, их необходимо сохранить.

ПРИМЕЧАНИЕ

Среду Turbo Pascal можно настроить таким образом, чтобы при ее запуске автоматически происходило создание нового файла. Для этого следует сначала создать вышеуказанным образом новый файл (**File → New**), а затем в главном меню выбрать **Options → Save Turbo.Tr**.

СОХРАНЕНИЕ ФАЙЛА С ТЕКСТОМ ПРОГРАММЫ

Сохранять файл с текстом программы рекомендуется не только в конце разработки программы, но по мере ее создания. Чтобы произвести сохранение следует в главном меню выбрать **File → Save** или нажать клавишу «F2» на клавиатуре. При этом на экране появится диалоговое окно **Save file as (Сохранение файла)**, в котором вам нужно будет указать место сохранения и имя, которое будет носить файл (рис. 2.5).

Предлагаемые по умолчанию имена NONAME00.PAS, NONAME01.PAS и т.д., рекомендуется заменять на более осмысленные, хоть как-то отражающие назначение программы. Например, если программа является своего рода калькулятором, то и назвать ее следует чем-то вроде CALC. В противном случае, при использовании ничего не значащих имен типа NONAME00.PAS будет сложно в них ориентироваться.

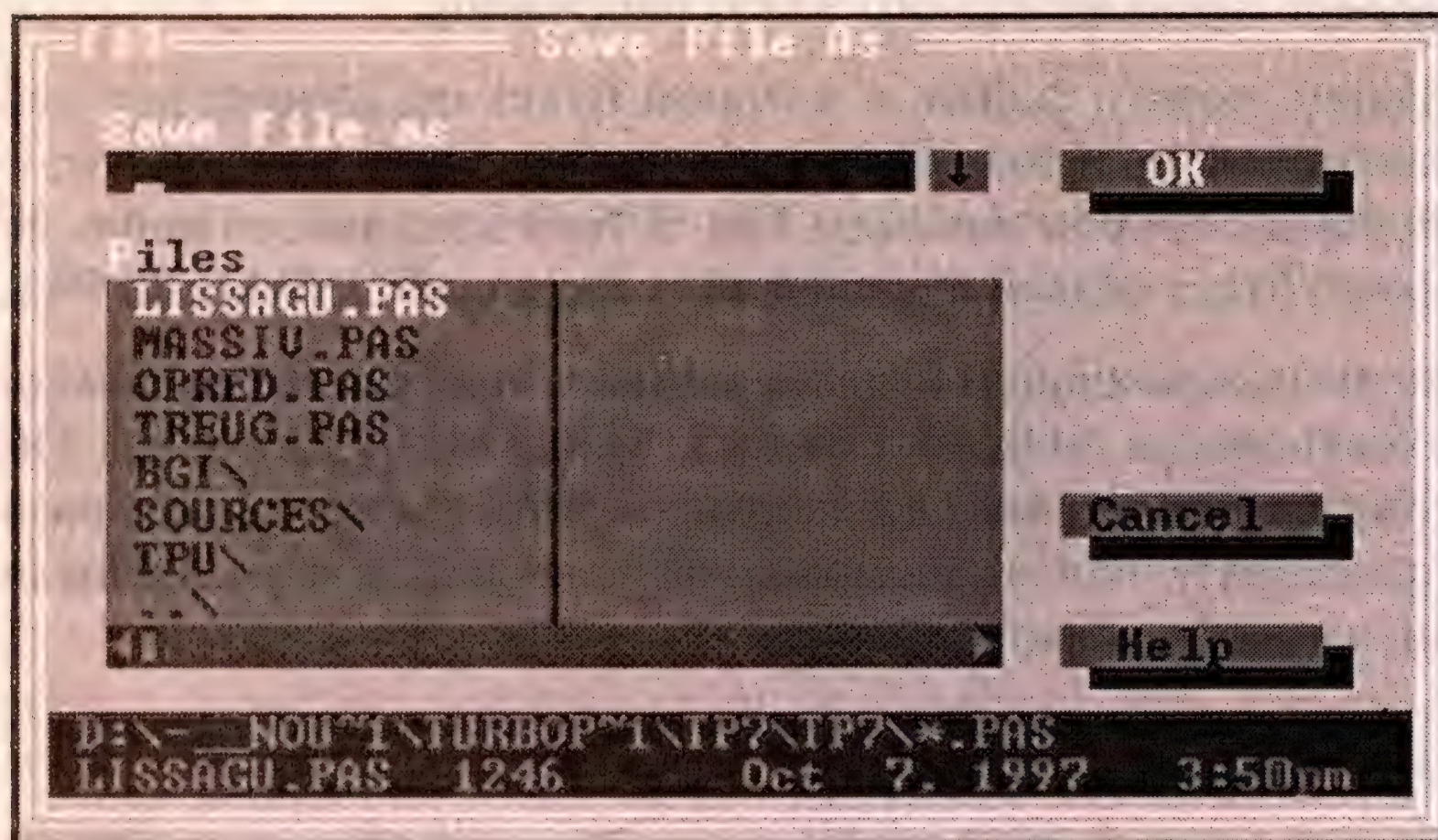


**Рис. 2.5. Диалоговое окно
Save file as (Сохранение файла)**

Поскольку среда Turbo Pascal является приложением, работающим под операционную систему MS-DOS, то имя файла не может превышать 8 символов (+3 символа отводится на расширение файла, которое автоматически устанавливается .PAS).

ПРИМЕЧАНИЕ

При запуске интегрированной среды Turbo Pascal из Windows сначала запускается сеанс MS-DOS, а уже в нем запускается среда.

Имена файлов могут содержать буквы латинского алфавита и цифры. Запрещено использование в именах файлов пробелов и символов:

* + = : ; [] { } < > ? /

Кроме того не рекомендуется использование букв русского алфавита (кириллицы) в именах файлов.

Диалоговое окно **Save file as (Сохранение файла)**, представленное на рис. 2.5, имеет стандартное для таких окон строение и содержит следующие элементы:

- Поле **Save file as**, в котором производится ввод имени файла, под которым он должен быть сохранен.
- Область **Files**, в которой представлен список каталогов (папок) и файлов с другими паскалевскими программами (с расширением .pas), содержащихся в том каталоге (папке), в который вы собираетесь производить сохранение. В случае необходимости вы можете переходить по разным каталогам (папкам) с целью выбора другого места для сохранения – делается это двойным щелчком мыши по имени нужного каталога (папки). Для перехода «вверх» в родительский каталог (папку), в котором содержится текущий каталог (папка) используется обозначение **..**
- Стандартные кнопки: **OK** – подтверждение сохранения и закрытие окна **Save file as (Сохранение файла)**, **Cancel** – отмена операции, **Help** – получение помощи.

Перемещение между различными элементами окна можно осуществлять либо мышью, либо с помощью клавиши «Tab».

Сохранять требуется также уже имеющийся на диске файл, если в нем были произведены какие-то изменения. В противном случае все поправки, произведенные с момента последнего сохранения файла, будут утеряны. При сохранении уже имеющегося файла после редактирования следует также выбрать **File → Save** или нажать клавишу «F2» на клавиатуре. При этом диалоговое окно **Save file as** уже больше появляться не будет.

При попытке закрыть несохраненный/измененный файл средой Turbo Pascal будет выдано предупреждающее сообщение (см. рис. 2.6). Вам будет предложено либо сохранить файл при закрытии – нажать **Save**, либо не сохранять файл при закрытии – нажать **No**, либо вообще отказаться от закрытия файла – нажать **Cancel**.

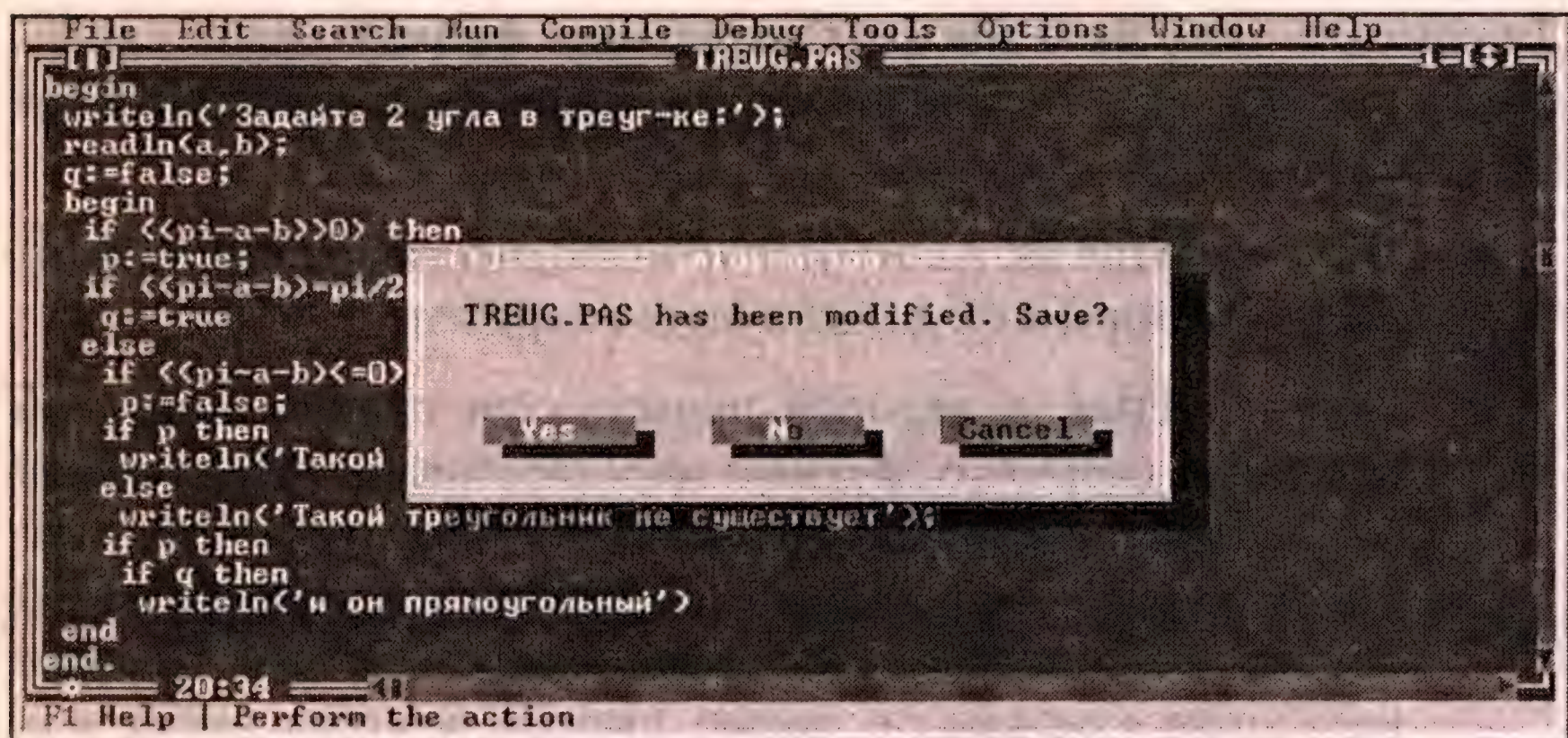


Рис. 2.6. Предупреждение

СОЗДАНИЕ ТЕКСТА ПРОГРАММЫ. РАБОТА С ТЕКСТОВЫМ РЕДАКТОРОМ СРЕДЫ TURBO PASCAL

Текстовый редактор, встроенный в среду Turbo Pascal, предоставляет разработчику достаточно развитые возможности создания и редактирования текстов программ. О том, что вы находитесь в режиме текстового редактора говорит наличие курсора — небольшого мигающего прямоугольника — в окне с текстом программы.

Если среда находится в другом состоянии — в режиме выбора пункта главного меню, то курсор исчезает, а в строке меню появляется цветной указатель, выделяющий один из пунктов. Переход от режима главного меню в режим текстового редактора и наоборот осуществляется либо с помощью мыши, либо с помощью клавиш «F10» — для перехода в режим выбора пункта главного меню — и «Esc» — для возврата в режим текстового редактора.

Текст программы вводится с клавиатуры, как это делается в обычных текстовых редакторах. При этом служебные слова, являющиеся элементами языка автоматически подсвечиваются желтым цветом, тогда как основной текст программы отображается зеленым. Благодаря этому разработчик уже на этапе набора текста

программы может оперативно отслеживать возможные ошибки в ключевых словах.

ПРИМЕЧАНИЕ

Это при стандартных настройках интегрированной среды Turbo Pascal. В принципе предусмотрена возможность изменения цветового оформления отображения программы выбрав в главном меню **Options → Environment → Colors**.

По окончании набора в текущей строке следует нажать на клавишу «Enter», чтобы перейти на следующую строку. Необходимо иметь в виду, что компилятор Turbo Pascal может воспринимать только те строки программы, которые не превышают 126 символов. В противном случае будет выдано сообщение об ошибке.

Перемещение по тексту программы в окне осуществляется стандартными средствами, предусмотренными в большинстве текстовых редакторов.

Так прокручивать текст программы можно либо с помощью полос прокрутки, расположенных по краям окна программы, либо с помощью клавиш «PgUp» и «PgDn» — пролистывать постранично текст программы вверх или вниз.

Кроме того, поддерживается работа следующих клавиш и клавиатурных сочетаний:

- «Home» — переход в начало текущей строки.
- «End» — переход в конец текущей строки.
- «Ctrl»+«PgUp» — переход в начало текста программы.
- «Ctrl»+«PgDn» — переход в конец текста программы.
- Клавиши-стрелки — для навигации по тексту.

Если при вводе допускается какая либо ошибка, то предыдущий символ можно стереть с помощью клавиши Backspace. Чтобы стереть следующий за курсором символ, следует воспользоваться

клавишей «Del». Нажатие сочетания клавиш «Ctrl»+«Y» приводит к стиранию всей строки, в которой находится курсор.

В нормальном режиме текстовый редактор среды Turbo Pascal 7.0 находится в режиме вставки, то есть набираемые символы раздвигают уже имеющийся в строке текст. Если же нажать на клавишу «Ins» то будет осуществлен переход в режим замещения, когда набираемые символы будут замещать собой имеющийся, встречающийся по пути текст. В этом смысле все организовано абсолютно так же, как и в любом другом текстовом редакторе, например в том же Microsoft Word.

Курсор в режиме замещения принимает вид большого прямоугольника размером с букву. Обратно перейти из режима замещения в режим вставки можно повторным нажатием клавиши «Ins» на клавиатуре.

ПРОВЕРКА И ОТЛАДКА ТЕКСТА ПРОГРАММЫ

По мере работы над программой следует периодически нажимать клавишу «F9», благодаря чему будет производиться проверка текста программы на наличие синтаксических ошибок. В случае нахождения какой-либо ошибки внизу окна будет выдано соответствующее сообщение, в котором будет указан номер ошибки и приведено краткое пояснение. Описание возможных сообщений об ошибках приведено в конце данной книги.

После завершения набора текста программы следует попробовать ее исполнить. Для этого следует нажать сочетание клавиш «Ctrl»+«F9» или в главном меню выбрать **Run → Run**. В результате программа будет откомпилирована, связана (в случае необходимости) с библиотеками стандартных процедур, загружена в оперативную память и запущена. Если на каком-то из данных этапов будет выявлена ошибка, то процесс будет прерван с соответствующим сообщением.

Ошибка может возникнуть и по ходу работы программы, например, если программе были переданы неправильные данные – как

если программа вычисляет квадратный корень из вводимого с клавиатуры числа, а вы передали ей отрицательное значение. В данном случае программа завершает свое выполнение, а в тексте программы подсвечивается красным та строка, на которой произошла ошибка.

В ходе отладки программы можно воспользоваться несколькими возможностями встроенного отладчика для поиска возможных проблем и их решения:

- **Можно выполнять программу по шагам.** Так, находясь в режиме редактирования текста программы, нажатие клавиши «F8» приводит к пошаговому выполнению строк программы. При этом если в строке есть обращение к процедуре или функции, то она будет исполнена без отслеживания ее работы. В случае же использования клавиши «F7» программа также будет выполняться по шагам, но при вхождении в процедуру или функцию все они также будут выполняться по шагам.
- **Можно установить точку останова, чтобы программа выполнялась лишь до определенного места.** Это позволит по отдельности проверять разные части программы. Установить точку останова можно, поместив в нужную строку курсор и нажав комбинацию клавиш «Ctrl»+«F8» (или выбрав в главном меню **Debug** → **Breakpoints**). При этом на экране появится диалоговое окно, в котором можно будет подрегулировать параметры останова – в поле **Condition** ввести дополнительное условие, по которому должен осуществляться останов, а в поле **Pass count** – число раз, которое должна выполняться строка программы, в которой установлена точка останова. По умолчанию условия никакого нет, число установлено 0 и программа просто завершает свое выполнение по достижении данной строки. Чтобы убрать установленную ранее точку останова, следует поместить курсор в строку останова и повторно нажать «Ctrl»+«F8» (или выбрать в главном меню **Debug** →

Breakpoints). В результате появится диалоговое окно управления всеми точками останова в программе, в котором следует выбрать одну из них и указать, что с ней делать: **Delete** – удалить, **View** – перейти к этой точке останова в программе, **Edit** – изменить параметры точки останова.

- **Можно по ходу выполнения программы просматривать значения переменных.** Для этого нужно интересующие переменные добавить в список просматриваемых переменных **Watches**. Сделать это можно установив курсор в области переменной и нажав сочетание клавиш «Ctrl»+«F7» (или выбрав в главном меню **Debug** → **Add watch**). В результате будет открыто диалоговое окно, в котором будет предложено ввести имя переменной, значение которой необходимо отслеживать. Список просматриваемых переменных располагается в отдельном окне, находящемся внизу среды Turbo Pascal.

ПРОСМОТР РЕЗУЛЬТАТОВ ВЫПОЛНЕНИЯ ПРОГРАММЫ

Зачастую пишутся программы, которые в результате своего выполнения выводят определенные значения и прекращают свое действие. При этом производится автоматический переход в интегрированную среду Turbo Pascal.

Чтобы вернуться к просмотру результатов выполнения программы следует нажать сочетание клавиш «Alt»+«F5» или в строке меню выбрать **Debug**→**User screen**. Нажатие на любую клавишу возвращает в режим интегрированной среды.

СОЗДАНИЕ ИСПОЛНЯЕМОГО ФАЙЛА ПРОГРАММЫ

Написанная программистом программа хранится в файле с расширением **.PAS**. Данные файлы по своей сути являются текстовыми и предназначены лишь для хранения исходных текстов программ. Запуск такого файла не приводит к запуску программы. Чтобы запустить программу по ее **.PAS**-файлу, сначала необходимо бу-

дет запустить среду Turbo Pascal, открыть в ней данный файл и уже оттуда запустить программу на выполнение указанным выше способом.

Однако существует возможность создания исполнимого файла программа, по которому программа может быть запущена без использования среды.

Чтобы создать такой файл (а он будет иметь расширение .exe – стандартное расширение исполняемого файла) следует нажать клавишу «F9» или выбрать в строке меню **Compile** → **Make**. В итоге исполнимый файл программы будет создан, о чем будет выдано соответствующее сообщение (см. рис. 2.7) с просьбой нажать любую клавишу для продолжения. Запуск исполняемого файла производится стандартными средствами, предусмотренными в Windows, например двойным щелчком мыши по нему.

Чтобы исполняемый файл был размещен на диске, необходимо, чтобы в разделе меню **Compile** напротив пункта **Destination** стояло значение **Disk**. Если же там стоит значение **Memory**, то файл будет создан в оперативной памяти и воспользоваться им вы не сможете. Поменять значение **Memory** на **Disk** можно простым вы-

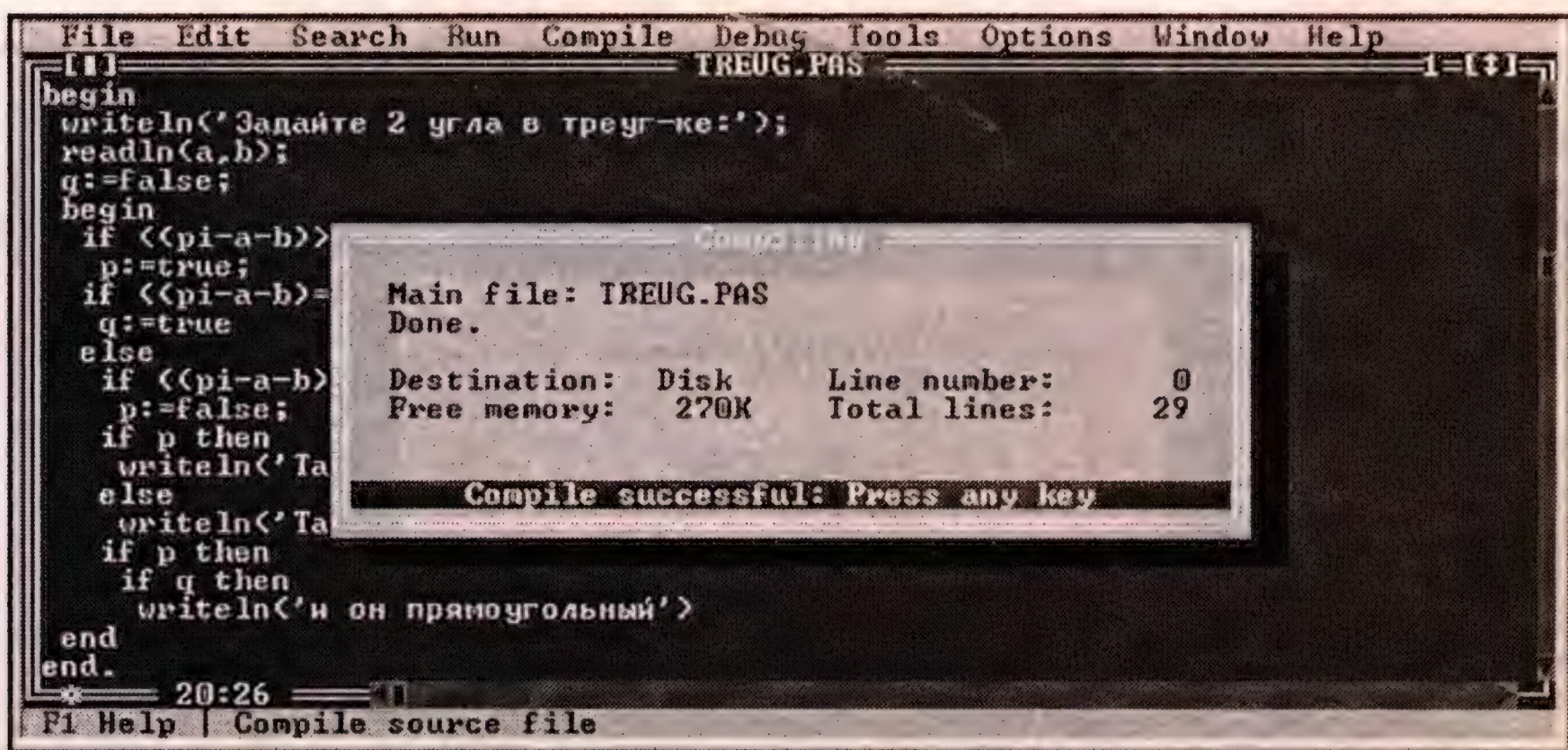


Рис. 2.7. Создание исполняемого файла программы прошло успешно. Для продолжения нужно нажать любую клавишу

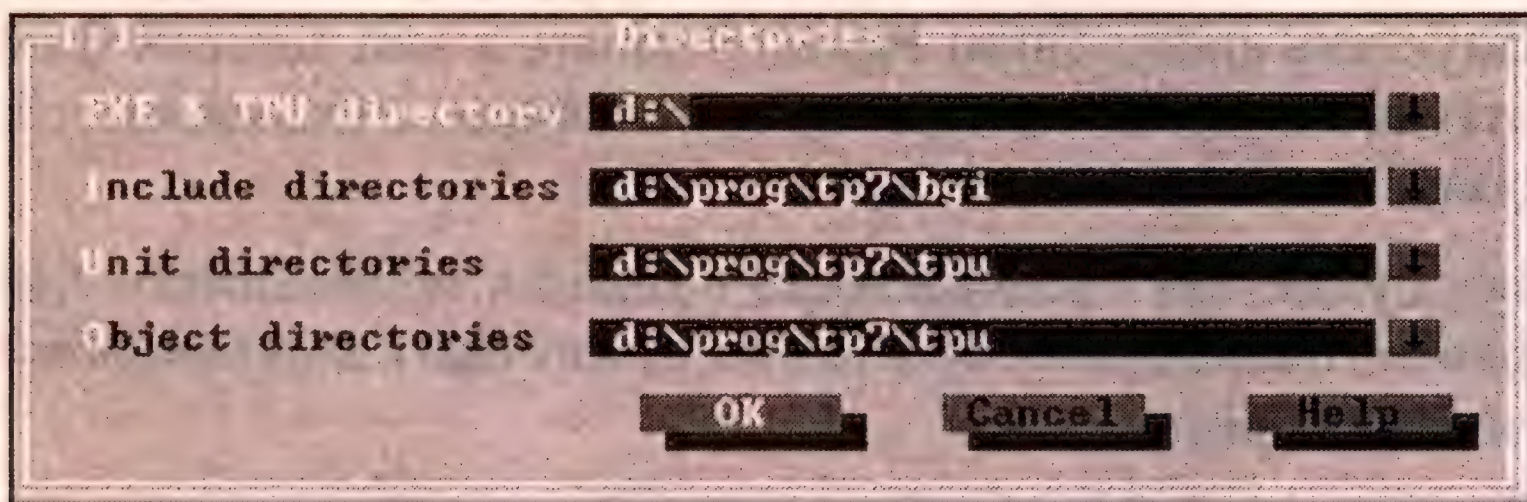


Рис. 2.8. Диалоговое окно, в котором задается каталог (папка) для помещения в него создаваемых исполняемых файлов

бором данного пункта. Повторный выбор меняет **Disk** на **Memory** и т.д.

Место на диске, куда должен будет помещен созданный исполняемый файл программы задается в настройках, вызываемых выбором в строке меню команды **Options** → **Directories**. При этом появляется диалоговое окно (см. рис. 2.8), в строке **EXE & TPU directory** которого как раз и задается место размещения создаваемых исполняемых файлов.

Необходимо обратить внимание, что если в строке **EXE & TPU directory** указан несуществующий каталог (папка), то создание

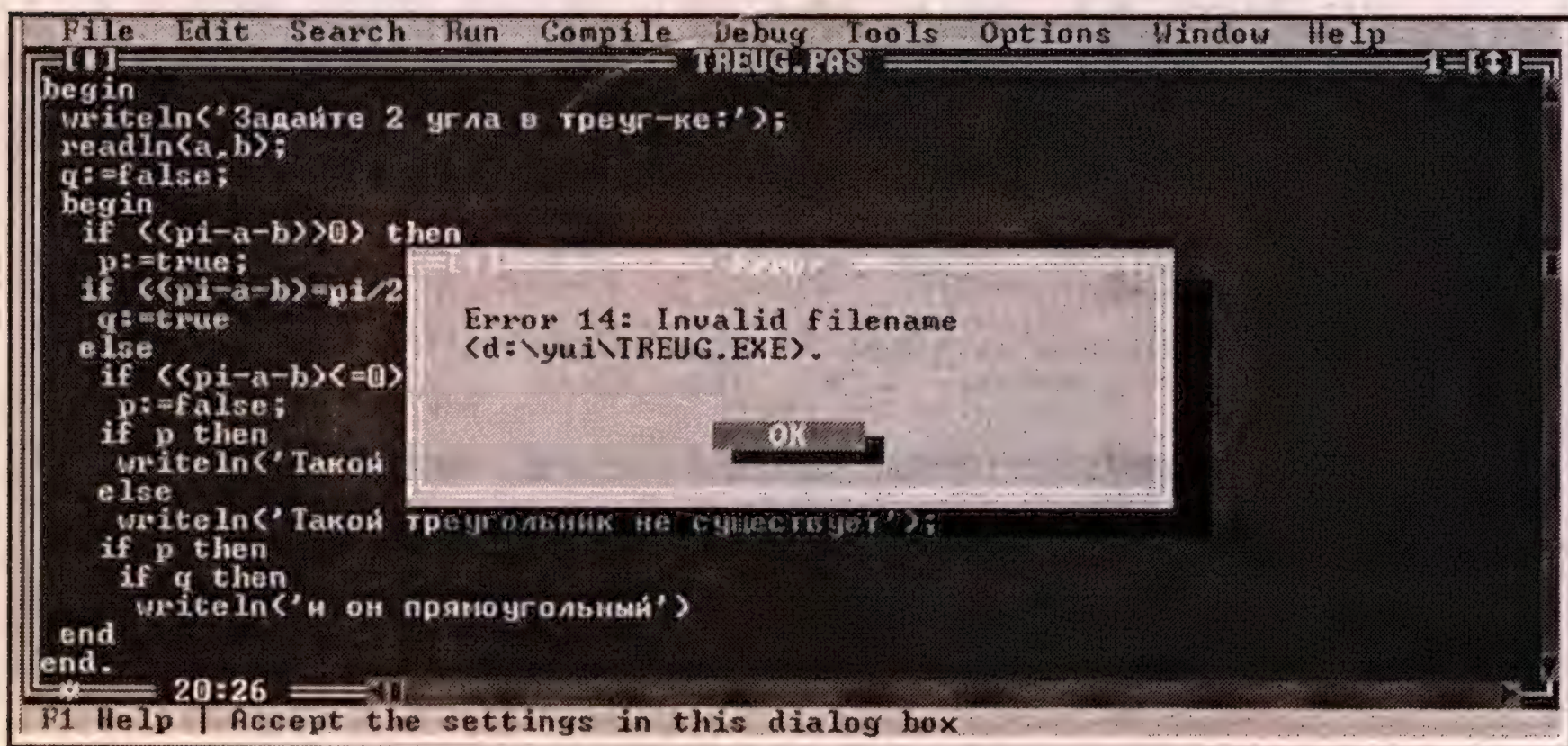


Рис. 2.9. Сообщение о несуществующем каталоге (папке), в который предполагается сохранение создаваемого исполняемого файла.

исполняемого файла программы будет прервано с выдачей соответствующего сообщения (рис. 2.9).

ПРИМЕЧАНИЕ

Если было произведено изменение в строке **EXE & TPU directory** и был указан новый каталог (папка), то, чтобы сохранить произведенные изменения для последующего использования, а не только на период текущего сеанса работы со средой Turbo Pascal, следует в главном меню выбрать **Options → Save TURBO.TP**.

2.5. ПРАКТИЧЕСКИЙ ПОШАГОВЫЙ ПРИМЕР СОЗДАНИЯ ПРОГРАММЫ

Далее в этом параграфе приведен пошаговый пример создания программы, ее сохранения, отладки и запуска, наглядно иллюстрирующий всю эту процедуру:

1. Запустите интегрированную среду двойным щелчком мыши по файлу `turbo.exe`.
2. Проследите, чтобы был создан новый файл с именем `NO-NAME00.PAS`. Если этого не произошло автоматически – сделайте это сами, выбрав в строке меню команду **File → New**.
3. Наберите текст программы, как показано на рис. 2.10. При этом проследите, чтобы ключевые слова `Program`, `Begin` и `End` по завершении своего набора меняли свой цвет – это значит они набраны без ошибок. Если цвет какого либо из этих слов не изменился, видимо, оно набрано неправильно. После ключевого слова `End` должна стоять точка, а после названия программы и команды `write` – точка с запятой.
4. Сохраните программу, нажав клавишу «F2» и введя имя файла, в котором должна храниться программа. В качестве имени укажите `PRIMER1.PAS`. Отметьте, что после сохранения файла на диске, в верхней части окна, в котором он открыт, появится его имя.

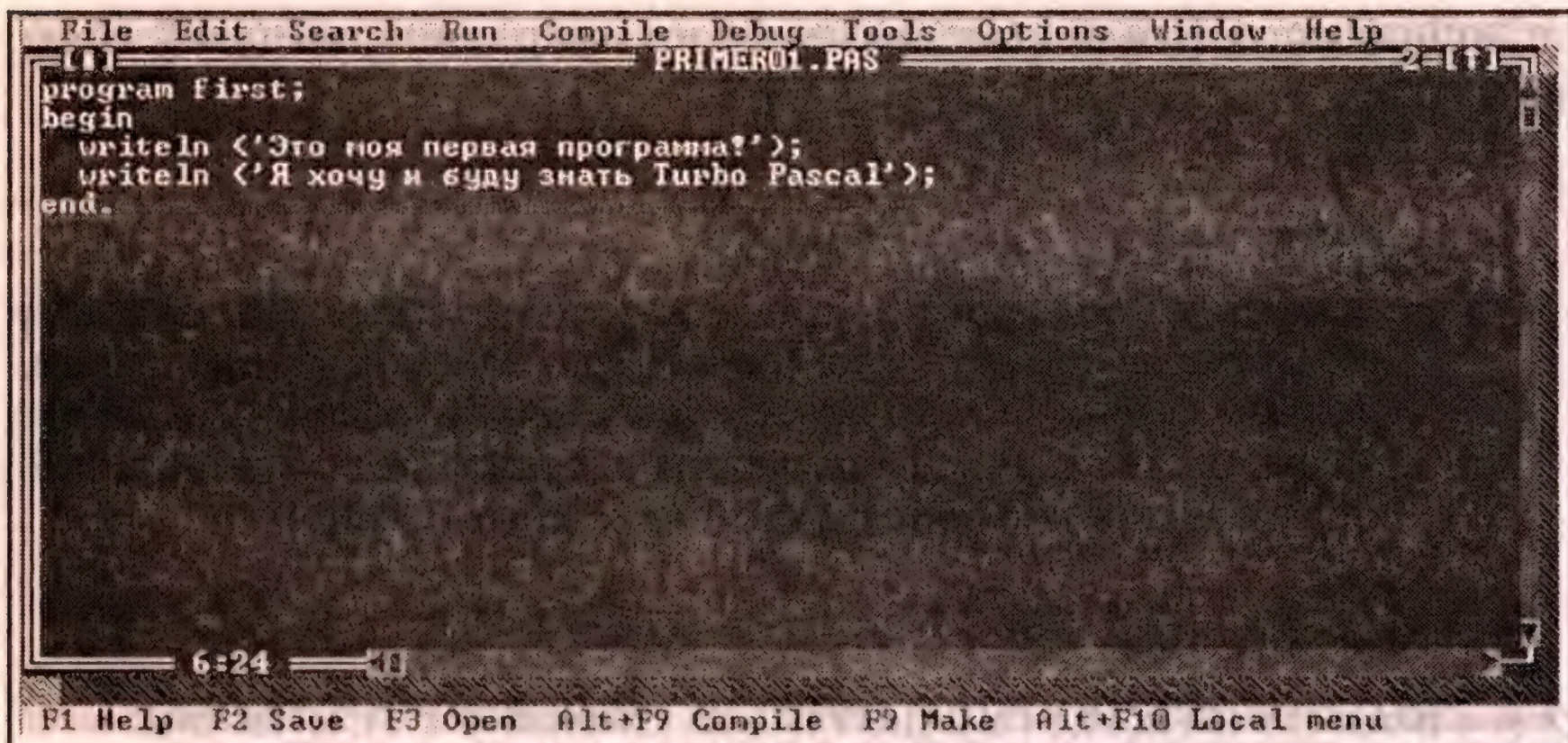


Рис. 2.10. Первая программа – первые шаги

5. Проверьте программу на наличие синтаксических ошибок. Для этого нажмите клавишу «F9». Если будут выявлены какие-либо ошибки – исправьте их. Теперь можно запустить программу. Данное действие производится с помощью выбора пункта главного меню **Run**→**Run** или нажатием комбинации клавиш «Ctrl»+«F9». Если программа набрана неправильно, то на экране появится сообщение компилятора об ошибке.

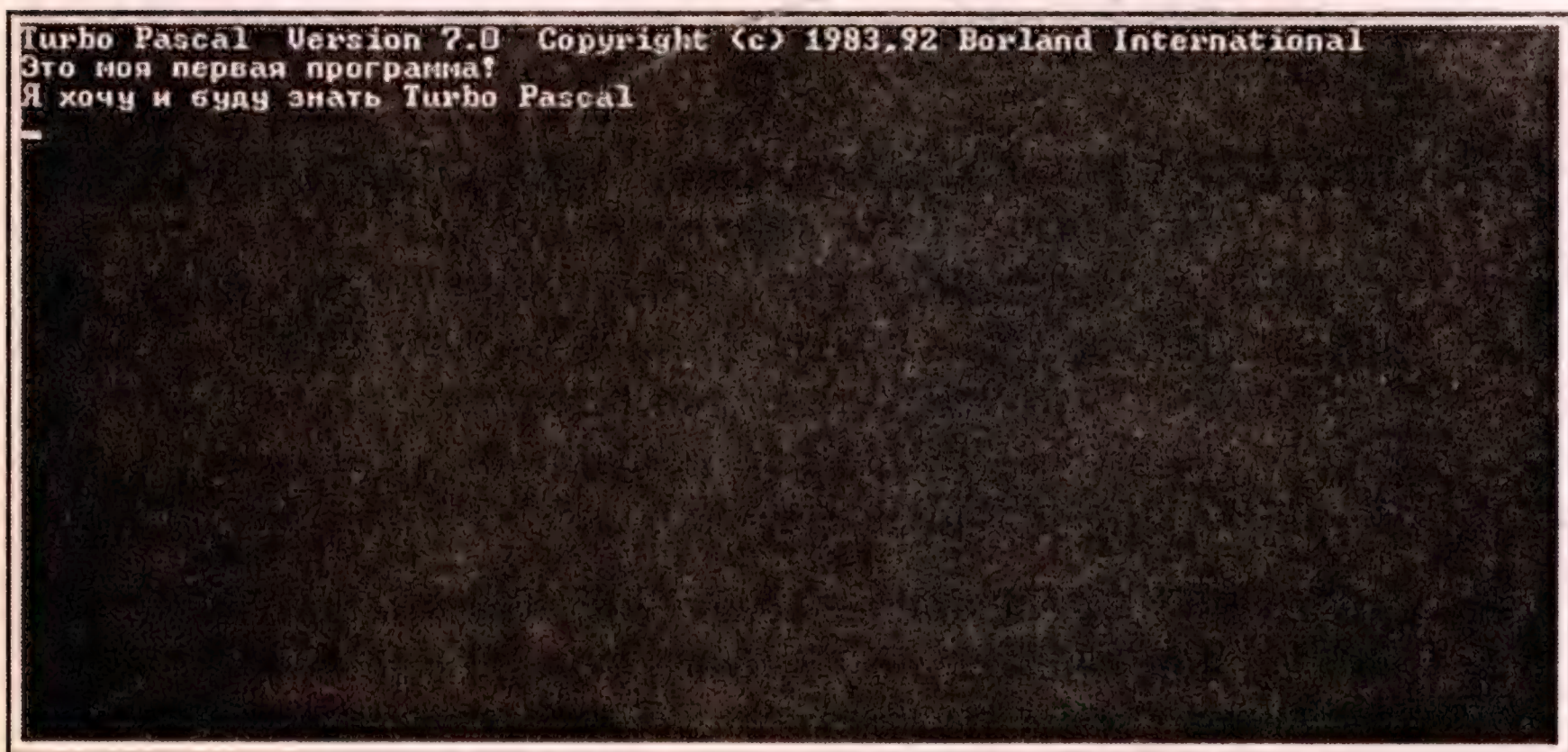


Рис. 2.11. Результат выполнения первой программы

6. Просмотрите результат выполнения программы, нажав сочетание клавиш «Alt»+«F5». Вернитесь в режим редактирования программы, нажав любую клавишу.

ШПАРГАЛКА. КОНТРОЛЬНЫЕ ВОПРОСЫ И ОТВЕТЫ ПО ТЕМЕ



ЧТО ТАКОЕ ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТЧИКА И КАКОВО ЕЕ НАЗНАЧЕНИЕ?

Интегрированная среда разработчика представляет собой набор инструментальных средств, которым снабжается язык программирования, и предназначена для повышения удобства и скорости разработки программ на соответствующем языке программирования.

Интегрированная среда разработчика в обиходе называется просто средой и включает в себя следующие основные элементы:

- текстовый редактор, адаптированный для разработки программ на данном языке (например, подсвечивающий другим цветом ключевые слова и т.п.);
- встроенный компилятор;
- встроенный отладчик, предназначенный для отладки программ.

Один и тот же язык программирования может иметь несколько сред разработчика от разных фирм-производителей.



КАК ПРОВЕРИТЬ ПРОГРАММУ НА НАЛИЧИЕ ОШИБОК?

Среда Turbo Pascal позволяет производить автоматическую проверку ошибок. Для этого следует нажать клавишу «F9». Однако таким способом могут быть обнаружены только лишь синтаксические, базовые ошибки в программе (например, неправильно написано ключевое слово, отсутствует точка с запятой и т.п.). Логические ошибки таким образом отслежены быть не могут.



КАК ИЗ СРЕДЫ TURBO PASCAL ЗАПУСТИТЬ ПРОГРАММУ НА ВЫПОЛНЕНИЕ?

Чтобы запустить программу, текст которой отображается в текущем активном окне, следует либо в строке меню выбрать Run → Run, либо нажать сочетание клавиш «Ctrl» + «F9».



КАК ПРОСМОТРЕТЬ РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ПРОГРАММЫ?

Если программа выполняет какие-либо действия и завершает свою работу выводом результатов, то после завершения программы будет автоматически осуществлен переход в интегрированную среду. Чтобы вернуться к просмотру результатов выполнения программы, следует нажать сочетание клавиш «Alt» + «F5».

3.1. ОСНОВНЫЕ ЭЛЕМЕНТЫ ПРОГРАММЫ И АЛФАВИТ ЯЗЫКА

Итак, программа, написанная на любом языке программирования, представляет собой набор команд, “понятных” компилятору, и параметров, необходимых для выполнения этих команд (некоторые команды могут использоваться без параметров).

Основными элементами программы являются:

- **Операторы** — команды, определяющие структуру программы (например, операторы ветвления и операторы цикла) или выполняющие какие-либо действия (например, арифметические операции или операции сравнения).
- **Ключевые слова** — команды, используемые обычно для отделения одних частей программы от других. Ключевые слова предназначены для компилятора, теряются в процессе построения исполняемой программы и не выполняются в прямом смысле этого слова.
- **Директивы компилятору** — аналогичны ключевым словам, но воздействуют не на программу и ее структуру, а на процесс компиляции и построения программы. Часть директив компилятору можно установить с помощью главного меню интегрированной среды или параметров компилятора командной строки, однако использование директив в тексте программы помогает использовать их

избирательно, не для всей программы, а только для отдельных ее частей.

- **Переменные и константы** — внутренние структуры программы, предназначенные для хранения, перемещения и других видов работы с информацией.
- **Процедуры и функции** — команды языка программирования, встроенные в него или добавленные разработчиком для приобретения программой необходимой функциональности. Последовательность выполнения процедур и функций, то есть структура программы, определяется операторами и ключевыми словами, а характер выполнения одних и тех же процедур и функций в различных ситуациях задается переменными и константами.

Все вышеперечисленные элементы программы имеют свои уникальные имена, называемые идентификаторами, которые и используются при написании программы. Например, один из операторов ветвления (условный оператор) практически во всех языках программирования имеет имя `if`, арифметическая операция сложения обозначается символом `+`, переменные и константы имеют имена, определяемые разработчиком, и так далее.

Для назначения имен элементам программы (то есть для создания идентификаторов) используются следующие символы:

- латинские алфавитные строчные и прописные: от “a” до “z” и от “A” до “Z”, а также символ подчеркивания “_” и цифры (но только не в начале имени);
- знаки арифметических операций: “+” (сложение), “-” (вычитание), “*” (умножение), “/” (деление);
- символы логической и арифметической группировки: “(“ и “)”;

Существует еще множество других специальных символов, рассмотрение которых в данном месте преждевременно. Их назначение будет объяснено ниже, в соответствующих разделах. Здесь же надо отметить одно правило, которое следует запомнить: все команды в Turbo Pascal должны заканчиваться точкой с запятой.

Правила формирования имен элементов программы будет рассмотрено в разделе, посвященном переменным.

3.2 СТРУКТУРА ПРОГРАММЫ

Программа на языке Turbo Pascal состоит из разделов, каждый из которых начинается специальным ключевым словом, характерным для конкретного раздела:

Program	Заголовок программы
Uses	Раздел подключения модулей — библиотек дополнительных процедур и функций
Label	Раздел описания меток безусловного перехода
Const	Раздел описания констант
Type	Раздел описания типов данных
Var	Раздел описания переменных
Procedure	Раздел описания подпрограмм — процедур и функций, содержащих некоторые элементы логики программы
Function	
Begin	Начало раздела описания логики программы
	Раздел описания логики программы
End.	Конец раздела описания логики программы

Любую программу можно представить в виде двух частей: командной, описывающей логику программы (то есть действия, которые могут выполняться программой), и структуры данных, с которыми работает командная часть.

Структура данных описывается с помощью разделов описания констант, типов данных и переменных, начинающихся, соответс-

твенно с ключевых слов `Const`, `Type` и `Var`. Командная часть представлена разделом описания логики программы, разделом описания процедур и функций, а также разделом описания меток безусловного перехода `Label`. Подключаемые модули представляют собой набор процедур, функций и данных, с которыми они работают. Поэтому при подключении модуля к программе, изменяется и структура данных программы, и командная часть.

Практически все вышеперечисленные разделы программы не являются обязательными. Более того, использование раздела `Label`, например, вообще не желательно. Единственным обязательным разделом является раздел описания логики программы, в котором реализуется алгоритм решения поставленной задачи. Данный раздел должен начинаться ключевым словом `Begin` и заканчиваться ключевым словом `End` с точкой после него. Если такого раздела в тексте программы нет, то она не будет восприниматься как правильная. Если же такой раздел в программе есть, то все остальные могут и отсутствовать.

Таким образом, простейшая программа на языке Turbo Pascal выглядит как:

```
Begin  
End.
```

Правда смысла в такой программе мало – она ничего не делает вообще – но она является правильной рабочей программой, которая вполне корректно компилируется и выполняется.

Несмотря на необязательность раздела `Program`, все-таки рекомендуется включать заголовок во все программы. Заголовок программы не используется в процессе выполнения программы и вообще теряется в процессе ее компиляции и линковки, но зато позволяет при открытии исходного текста быстро определить, что эта программа делает. В данном разделе должно присутствовать всего одно слово – название программы. Правила составления имен вообще и для программы в частности описаны ниже при обсуждении переменных и констант.

Таким образом простейшая программа приобретает следующий вид:

```
Program Primer;  
  Begin  
  End.
```

3.3. ПЕРЕМЕННЫЕ

Программа создается для того, чтобы работать с какой-либо информацией. С точки зрения программы, все данные, используемые в ней, можно разбить на три группы:

- **входные данные** — получаемые программой извне: информация, вводимая пользователем, полученная по модему, от сканера или каких-либо других устройств ввода информации;
- **выходные данные** — выдаваемые программой в результате ее работы: изображение на экране или принтере, отправка данных по модему, сохранение файлов на дисках;
- **внутренние данные** — хранимые программой в оперативной памяти с целью преобразования входных данных в выходные. Следует заметить, что не все входные данные, полученные программой, станут внутренними. Например, при вводе пароля пользователем, введенная строка является входными данными, однако, если введен неверный пароль, то программа не будет размещать данную строку в памяти.

То же самое можно сказать и про выходные данные. Программа, выдающая квадраты чисел, хранящихся в качестве внутренней информации, необязательно должна хранить еще и квадраты этих чисел, а может просто вычислять их при выводе.

ЧТО ТАКОЕ ПЕРЕМЕННАЯ. ПРАВИЛА ИМЕНОВАНИЯ ПЕРЕМЕННЫХ В TURBO PASCAL

Все внутренние данные программы хранятся в специальных структурах, называемых переменными. Переменная — это поименованная область оперативной памяти. Таким образом, программа обращается к переменным по их именам (идентификаторам), задаваемым программистом на этапе разработки программы. В Turbo Pascal используются следующие правила назначения имен:

- Имена переменных должны представлять собой одно слово, состоящее из строчных или прописных латинских символов, причем строчные и прописные символы не различаются между собой. Также, допустимо использование символа “подчеркивание” (“_”) и цифр в любом месте имени, за исключением его начала.
- Два разных элемента программы не могут иметь одинаковые имена, то есть имя, назначаемое переменной должно быть уникальным в пределах программы. В случае попытки описания, например, двух переменных с одинаковыми именами, компилятором будет выдана ошибка **Duplicate Identifier** (повторяющийся идентификатор). В скобках будет указано имя переменной, которая описывается повторно.
- Длина имени, назначаемого любому элементу программы, не должна превышать 63-х символов.

Рассмотрим примеры правильного составления имен переменных:

Peremennaja	Q	B52	Itreiuhi
My Peremennaja	Q12	B_12	UfhYsdk
My_ Peremennaja	L414	B12_	foRmila

Теперь рассмотрим примеры неправильного составления имен переменных:

- **Begin** — Является ключевым словом языка Pascal, поэтому имя данной переменной не удовлетворяет требованию уникальности в пределах программы.

- `begin` — Поскольку Turbo Pascal не различает строчных и прописных букв, это то же самое ключевое слово `Begin`, просто написанное с маленькой буквы.
- `Моя_переменная` — Имя содержит не латинские символы.
- `Name.` — В имени переменной используется недопустимый символ — точка.
- `My Peremennaja` — Имя представляет собой два слова.
- `_MyPeremennaja` — Имя начинается с символа подчеркивания “_”
- `44rurka` — Имя начинается с цифры.

Типы данных

Как уже было сказано чуть раньше, все данные, с которыми оперирует программа, хранятся в виде переменных — областей оперативной памяти, доступ к которым осуществляется по имени. При этом каждая переменная имеет свой тип — один из стандартных типов, предусмотренных в Turbo Pascal или созданный программистом.

Тип переменной характеризует:

- **Сколько памяти должно отводиться по переменную.** Как известно память компьютера условно представляет собой массив ячеек, в каждой из которых может быть помещена единица информации — байт. Один байт может содержать число от 0 до 255 (то есть 256 чисел, начиная с 0), 2 байта — число от 0 до 65535 ($256 \cdot 256 - 1$) и т.д. Так вот за разными типами закрепляется разное количество байтов, отводимых под переменные этих типов. Соответственно переменные разного типа имеют разные пределы тех чисел, которые могут в них содержаться.
- **Как должна восприниматься переменная.** Дело в том, что хотя компьютер и работает только с числовыми значениями, пользователь работает также с текстовыми и прочими. При этом чтобы, скажем, текстовые значения восприни-

мались компьютером, они должны быть переведены в числовую форму (в соответствии с определенными таблицами буквы заменяются на числа). Так вот, для того чтобы автоматически выполнялось преобразование числовой информации, хранимой в памяти, в нужный вид, при описании переменной компилятору необходимо указать, какого типа информация находится в данной переменной.

Итак, переменные описываются в разделе описания переменных, который начинается с ключевого слова `Var`. При этом указывается имя переменной и ее тип:

`Var`

`<Имя переменной> : <Тип переменной>;`

В качестве имени переменной можно использовать любое имя, соответствующее вышеприведенным правилам, а Тип переменной может быть следующим:

1. **Byte** — каждая переменная данного типа может содержать одно целое число в диапазоне от 0 до 255.
2. **Integer** — каждая переменная данного типа может содержать одно целое число в диапазоне от -32768 до +32767.
3. **Word** — каждая переменная данного типа может содержать одно целое число в диапазоне от 0 до +65535.
4. **LongInt** — каждая переменная данного типа может содержать одно целое число в диапазоне от -2147483648 до +2147483647.
5. **Real** — каждая переменная данного типа может содержать одно вещественное число в диапазоне от $-2.9 \cdot 10^{39}$ до $1.7 \cdot 10^{38}$. Точность хранения чисел — 11 знаков после запятой.
6. **Char** — значениями переменных данного типа являются одиночные символы. В любой момент выполнения программы в переменной данного типа может храниться только один символ.

- В языке Turbo Pascal имеется еще множество разных типов, о некоторых из них мы будем говорить ниже, а пока вполне достаточно восьми описанных. Рассмотрим примеры описания переменных:

Program MyPeremenna;	
Var	{начало раздела описания переменных}
A: Integer;	{описание переменной типа Integer с именем A}
Begin	{начало раздела описания логики программы}
End.	{конец раздела описания логики программы}

Данная программа не содержит выполняемых команд (раздел описания логики программы пуст), однако, в отличие от простейшей программы, она имеет потенциальную возможность работы с данными, структура которых описана в разделе описания переменных, начинающегося с ключевого слова `Var`. Структура данных состоит из одной переменной типа `Integer`, в которой в каждый момент времени при выполнении программы может находиться одно целое число в диапазоне от -32768 до $+32767$ (см. описание типа данных `Integer`). Теперь рассмотрим более сложный пример:

```

Program MyPeremen;
Var
    {начало раздела описания переменных}
A: Integer;
    {описание переменной A типа Integer}
    V: Real;
    {описание переменной V типа Real}

```


<code> nur: String[70];</code>	{описание переменной типа String с именем nur и максимальным размером 70 символов}
<code> Name: String;</code>	{описание переменной типа String с именем Name и максимальным размером 255 символов}
<code> Ch: Char;</code>	{описание переменной типа Char с именем Ch}
<code> Index: Integer;</code>	{описание переменной типа Integer с именем Index}
<code>Begin</code>	{начало раздела описания логики программы}
<code>End.</code>	{конец раздела описания логики программы}

Существует возможность описать несколько переменных одного типа в одной строке. Такое описание выглядит следующим образом:

Var

<Имя переменной>, ..., <Имя переменной> : <Тип переменной>;

Рассмотрим еще один пример описания переменных:

<code>Program MyPeremennaja;</code>	{заголовок программы, имя программы — MyPeremennaja}
<code>Var</code>	{начало раздела описания переменных}
<code> a, b, c: Real;</code>	{описание трех переменных типа Real с именами a, b и c}
<code> x1, x2: Integer;</code>	{описание двух переменных типа Integer с именами x1 и x2}
<code> Flag: Boolean;</code>	{описание одной переменной типа Boolean с именем Flag}
<code>Begin</code>	{начало раздела описания логики программы}
<code>End.</code>	{конец раздела описания логики программы}

Оператор присваивания. Совместимость типов данных. Преобразование типов

Оператор присваивания позволяет придать некоторое значение переменной, предварительно описанной в разделе описания переменных **Var**. Оператор присваивания всегда находится в разделе описания логики программы или является составной частью про-

цедур и функций, состоит из двоеточия и знака “равно” и имеет следующий вид:

<Имя изменяемой переменной> := <Значение переменной>;

В качестве Имени переменной должно использоваться имя одной из переменных, описанных в разделе `Var`, а в качестве Значения переменной одно из следующих выражений:

- значение, соответствующее типу изменяемой переменной;
- выражение, результат вычисления которого, имеет тип, соответствующий типу изменяемой переменной;
- переменная, тип которой соответствует типу данной переменной.

С точки зрения оператора присваивания, типы Изменяемой переменной и Значения переменной считаются соответствующими друг другу, если Значение переменной входит в подмножество возможных значений Изменяемой переменной. Тем не менее в Turbo Pascal существуют специальные средства (процедуры), позволяющие преобразовывать одни типы данных в другие (подробнее о них читайте в следующих главах:

- **chr** — преобразует символьное значение в числовое;
- **ord** — преобразует числовое значение в символьное;
- **trunc** — преобразует вещественное число в целое путем отсечения дробной части;
- **round** — преобразует вещественное число в целое путем округления.

Рассмотрим примеры правильного и неправильного использования оператора присваивания на примере:

```
Program Pris;  
Var  
  A1, A2: Real;  
  I1, I2: Integer;
```



```
F: Boolean;
S: String;
Ch: Char;
Begin
A1 := 1.25;      {правильно}
A2 := 2.33;      {правильно}
A2 := 7;         {правильно, хотя и в памяти будет храниться
                  7.00}
A2 := A1;        {правильно}
I1 := 4;         {правильно}
I1 := 4.38;      {неправильно}
I2 := 10;        {правильно}
I1 := I2;        {правильно}
I1 := A2;        {неправильно}
S := 'Наука';    {правильно}
Ch := 'Z';       {переменная Ch типа Char может иметь
                  своим значением один символ}
Ch := 'Россия';  {неправильно}
Ch := S;         {переменная Ch типа Char не может иметь
                  своим значением строку символов}
S := 10;         {переменная S типа String не может иметь в
                  качестве своего значения число}

F := True;       {правильно}
F := 1548993;    {неправильно}
End.
```

В данном примере оператор присваивания использовался в простейшем виде, когда в качестве нового значения изменяемой переменной используется либо прямо задаваемое в тексте значение, либо значение другой переменной.

Необходимо отметить, что строковые (типа `String`) и символьные (типа `Char`) значения указываются в одинарных кавычках, значением же переменной в результате выполнения оператора присваивания будет строка (символ для переменных типа `Char`) без кавычек.

АРИФМЕТИЧЕСКИЕ И ЛОГИЧЕСКИЕ ВЫРАЖЕНИЯ

В качестве правой части (присваиваемого значения) могут быть использованы арифметические и логические выражения.

Арифметическое выражение — это выражение, состоящее из целочисленных или вещественных переменных и числовых значений, соединенных между собой арифметическими операторами сложения (+), вычитания (-), умножения (*) и деления (/). Также допустимо использование скобок. Приоритет одних арифметических операций над другими и изменение приоритетов при использовании скобок в арифметических выражениях соответствуют обычным математическим правилам.

При использовании арифметических выражений в качестве значений, присваиваемых переменным, необходимо следить за типом результата этих выражений. Например, при сложении двух переменных типа `Integer`, результирующее значение может выходить за пределы множества значений типа `Integer`:

```
Program PrimerPries2;
Var                                {начало раздела описания переменных}
  I1, I2, I3: Integer;             {описание трех переменных типа Integer с
                                   именами I1, I2 и I3}
Begin                              {начало раздела описания логики программы}
  I1 := 11111;                     {присвоение переменной I1 значения 11111}
  I2 := 30000;                     {присвоение переменной I2 значения 30000}
  I3 := I1 + I2;                   {попытка некорректного присвоения
                                   переменной I3 значения I1 + I2 = 11111 + 30000
                                   = 41111, выходящего за пределы множества
                                   типа Integer}
End.                               {конец раздела описания логики программы}
```

Еще одна особенность вычисления арифметических выражений состоит в том, что, если в нем используется деление, то результат выражения всегда является вещественным числом, то есть тип результата такого выражения будет `Real`, даже, если деление может быть произведено без остатка. Допустим имеются три переменный `A1`, `A2` и `A3` целочисленного типа `Integer`. Тогда:


```
A1 := 50;  
A2 := 10;  
A3 := A1 / A2;    {ошибка, так как 50 / 10 = 1.00, а это вещественное число  
                  (Real)}
```

Логические выражения — это выражения, состоящие из условий и логических переменных (имеющих тип Boolean), соединенных между собой логическими операторами AND (“И”), OR (“ИЛИ”) и NOT (“НЕ”). Значение такого выражения имеет тип Boolean. Условия, используемые в логических выражениях, состоят из переменных и значений, соединенных с помощью операторов условий “БОЛЬШЕ” (>), “МЕНЬШЕ” (<), “РАВНО” (=), “НЕ БОЛЬШЕ” (<=), “НЕ МЕНЬШЕ” (>=) и “НЕ РАВНО” (<>). Рассмотрим несколько примеров условий:

10 > 5	{Условие, значение True (истина)}
10 < 5	{Условие, значение False (ложь)}
20 >= 20	{Условие, значение True (истина)}
20 <= 20	{Условие, значение True (истина)}
20 <= 19	{Условие, значение False (ложь)}

Логический оператор AND для двух логических значений (значения, с которыми работают любые операторы, называют операндами) возвращает истинное значение (True), если оба логических значения истинны; если одно из значений ложно, то оператор AND возвращает значение False. Например, выражение True AND True является истинным (имеет значение True), а выражение True AND False — ложно. Следует заметить, что от перестановки операндов при использовании логических операторов результат выражения не меняется. Таким образом, выражение False AND True также ложно.

Оператор OR, также использующий два операнда возвращает истинное значение, в случае, если хотя бы один из операндов имеет истинное значение. Например, выражение True OR False является истинным, а выражение False OR False — ложно.

Логический оператор NOT работает с одним операндом и возвращает значение False, если операнд имеет истинное значение и принимает истинное значение, если операнд имеет значение

False. Таким образом, $\text{NOT True} = \text{False}$, а $\text{NOT False} = \text{True}$.

Также как и арифметические операторы, логические операторы имеют приоритеты, определяющие порядок выполнения операторов. Так, наивысшим приоритетом обладает оператор NOT, затем идет оператор AND, наименьший приоритет имеет оператор OR. Например, выражение $\text{NOT True AND False}$ является ложным, потому что сначала выполняется оператор NOT: $\text{NOT True AND False} = \text{False AND False} = \text{False}$. Если бы оператор AND выполнялся раньше, чем оператор NOT, то значение данного выражения было бы истинно: $\text{NOT True AND False} = \text{NOT False} = \text{True}$. Однако, это не так. Также, можно использовать круглые скобки для изменения порядка выполнения операторов: $\text{NOT (True AND False)} = \text{NOT False} = \text{True}$.

Теперь рассмотрим примеры логических выражений с использованием условий и логических операторов:

$(77 > 5) \text{ AND } (33 > 7)$ {Логическое выражение, состоящее из двух истинных условий: $(10 > 5)$ и $(10 > 7)$. Соответственно, по правилам работы оператора AND, результат выражения имеет значение True}

$(56 < 5) \text{ OR } (87 > 5)$ {Логическое выражение, состоящее из двух условий: истинного $(10 < 5)$ и ложного $(10 > 5)$. Соответственно, по правилам работы оператора OR, результат выражения имеет значение True}

$\text{NOT } (20 \geq 20)$ {Логическое выражение, состоящее из одного истинного условия. По правилам работы оператора NOT, значение данного выражения — False}

Механизм работы оператора присваивания

Работа оператора присваивания состоит из двух последовательных этапов:

1. Вычисление выражения, стоящего в правой части.

2. Присвоение переменной стоящей в левой части значения, полученного в пункте 1.

Таким образом, из-за того, что процесс присвоения состоит из двух отдельных этапов, в левой и правой части оператора присваивания может находиться одна и та же переменная. Рассмотрим пример:

```
Program Pris5;  
Var                               {начало раздела описания переменных}  
  I1: Integer;                   {описание переменной типа Integer с именем I1}  
Begin                             {начало раздела описания логики программы}  
  I1 := 35;                       {присвоение переменной I1 значения 18}  
  I1 := I1 + 12;                  {присвоение переменной I1 значения (I1 + 15). Таким  
                                образом, в результате выполнения данной команды,  
                                переменная I1 получает значение (I1 + 15) = 50 }  
End.                             {конец раздела описания логики программы}
```

Поясним кажущуюся некорректность выражения $I1 := I1 + 15$. С точки зрения математики, такое выражение действительно недопустимо, но данное выражение не является математическим — это команда языка программирования, выполняемая в два этапа, на первом из которых происходит вычисление выражения, стоящего в правой части: $I1 + 15 = 50$, а на втором — присвоение этого значения переменной $I1$. Таким образом, данное выражение вполне корректно.

Отметим, что вне зависимости от того использовался ли оператор присваивания для занесения значения в переменную, или нет, переменная все равно будет иметь какое-либо значение. Это связано с тем, что физически переменная представляет собой совокупность ячеек памяти, в которых не может ничего не храниться. Таким образом, рекомендуется в начале программы **инициализировать** все переменные — заносить в них некоторые начальные значения. Обычно в качестве начальных значений для числовых переменных (переменных типа Integer, Byte и т.д.) используется значение 0, а для строковых — пустая строка: ''.

3.4. КОНСТАНТЫ

Помимо переменных, для описания данных в программе используются постоянные выражения (константы) — выражения любого допустимого в языке программирования типа, определяемые программистом на этапе написания программы и не меняющиеся в процессе выполнения программы.

Постоянные выражения, также как и переменные, могут иметь имя, по которому используются во всех выражениях. Как и переменные, константы имеют тип данных, хотя обычно он задается не программистом, а определяется автоматически компилятором. Основным свойством постоянных выражений, отличающих их от переменных, является неизменность значения константы во время выполнения программы.

Постоянные выражения получают свое значение на этапе написания программы и не меняются в процессе выполнения программы. Более того, константы вообще теряются в процессе компиляции, и в исполняемой программе оказывается только значение постоянного выражения.

Константы описываются в разделе описания констант, начинающемся с ключевого слова `Const`. Описание константы имеет следующий вид:

`Const`

`<Имя константы> = <Значение константы>;`

В качестве Имени константы используется любое имя, удовлетворяющее правилам именования элементов программы, а в качестве Значения константы — любое выражение, тип которого допустим в Turbo Pascal, например арифметическое или логическое. При этом тип постоянного выражения указывать не нужно, так как компилятор может определить его самостоятельно. Рассмотрим примеры описания констант

```
Program MyConst;
```

```
Const
```

```
  cot = 10;
```



```
cot2 = 500*12;  
Radius = 1.0;  
Begin  
End.
```

В этом примере константам присваиваются простые значения, однако возможно описание констант, представляющих собой значения более сложных выражений (арифметических или логических), в том числе и содержащих в своем составе другие константы.

3.5. КОММЕНТАРИИ

Комментариями называются части программы, ограниченные фигурными скобками. Все комментарии игнорируются компилятором, поэтому могут содержать любой текст:

```
{ Комментарий }
```

Вместо фигурных скобок могут использоваться последовательности символов “(” для указания начала комментария и “)” для указания окончания комментария:

```
(* Комментарий *)
```

Комментарии могут занимать много строк или находиться в одной строке с командами программ:

```
Var  
  i: Integer; {Описание переменной для счетчика цикла}  
Begin  
  WriteLn; {Перевод строки}  
  { Вывод нескольких строк на  
    экран с использованием цикла }  
  For i := 1 To 10 Do  
    WriteLn('Строка номер ', i); (*Вывод строки-константы и  
    значения счетчика цикла*)  
End.
```


3.6. СТАНДАРТНЫЕ МАТЕМАТИЧЕСКИЕ ФУНКЦИИ TURBO PASCAL

Описание стандартных математических функций Turbo Pascal сведено в табл. 3.1.

Таблица 3.1. *Стандартные математические функции Turbo Pascal*

Функция	Типа аргумента	Тип результата	Описание
abs(x)	real, integer	real, integer	Модуль аргумента
ArcTan(x)	real	real	Арктангенс
cos(x)	real	real	Косинус, угол задается в радианах
exp(x)	real	real	Экспонента
frac(x)	real	real	Дробная часть числа
int(x)	real	real	Целая часть числа
ln(x)	real	real	Логарифм натуральный
pi	real	real	Число «пи»
random	real	real	Псевдослучайное число, взятое из интервала 0..1
random(x)	integer	integer	Псевдослучайное число, взятое из интервала 0..(x-1)
randomize	-	-	Инициализация генератора случайных чисел. Ее нужно произвести прежде чем воспользоваться функциями random и random(x)
sin(x)	real	real	Синус, угол задается в радианах
sqr(x)	real	real	Возводит число (аргумент) в квадрат
sqrt(x)	real	real	Извлекает корень из аргумента

3.7. АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ DIV И MOD

Помимо стандартных арифметических операций сложения, вычитания, умножения и деления, в Turbo Pascal предусмотрены еще два:

- **div** – целочисленное деление;
- **mod** – остаток от деления.

Целочисленное деление (**div**) отличается от обычной операции деления тем, что возвращает только целую часть частного, дробная же часть отбрасывается.

Таким образом, результатом такого деления всегда является целое число, тогда как результатом обычного деления всегда является вещественное число (даже если деление было выполнено без остатка и получилось, например, частное 2, но это будет не 2, а 2.0).

Перед выполнением целочисленного деления (**div**) и делимое и делитель округляются до целых значений. Результат целочисленного деления всегда равен нулю, если делимое меньше делителя.

Примеры:

$$11 \text{ div } 5 = 2$$

$$10 \text{ div } 3 = 3$$

$$2 \text{ div } 3 = 0$$

Деление по модулю восстанавливает остаток от целочисленного деления.

Например:

$$11 \text{ mod } 5 = 1$$

$$10 \text{ mod } 3 = 1$$

$$14 \text{ mod } 5 = 4$$

3.8. ПОРЯДОК ВЫПОЛНЕНИЯ ОПЕРАЦИЙ В ПРОГРАММАХ
НА TURBO PASCAL

Выполнение каждой операции происходит с учетом ее приорите-
та (см. табл. 3.2).

Таблица 3.2. Порядок выполнения операций

Операция	Приоритет
not	Первый (высший)
*, /, div, mod, and	Второй
+, -, or, xor	Третий
=, >, <, <>, >=, <=	Четвертый (низший)

Первыми выполняются операции с наивысшим приорите-
том. При этом если переменная или константа (операндами)
стоит между двумя операциями с разным приоритетом, то
выполняться будет сначала та, у которой приоритет выше.
Если же операции имеют равный приоритет, то выполняться
они будут в порядке очередности слева направо. Так же
как и в математике, порядок выполнения операций можно
регулировать с помощью использования круглых скобок
– сначала операции будут выполняться в них.

ШПАРГАЛКА. КОНТРОЛЬНЫЕ ВОПРОСЫ И ОТВЕТЫ ПО ТЕМЕ



ЧТО ТАКОЕ ПЕРЕМЕННАЯ?

Под переменной принято понимать область оперативной памяти, доступ к которой осуществляется по имени (идентификатору). То, какие данные хранятся или могут храниться в переменной, определяется типом переменной. В соответствии с типом под разные переменные (переменные разного типа) отводится разный объем оперативной памяти.



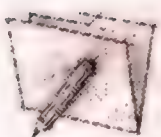
ЧТО ТАКОЕ ИДЕНТИФИКАТОР?

Идентификатор – это имя элемента программы: команды, переменной т.д. Соответственно по идентификатору (имени) осуществляется обращение к переменным, вызов команд и т.п.



ПЕРЕМЕННЫЕ КАКОГО ТИПА ПРЕДУСМОТРЕНЫ В TURBO PASCAL?

В языке TurboPascal предусмотрены следующие стандартные типы данных: целые числа (тип **Integer** – значения от -32768 до +32767, тип **Byte** – значения от 0 до 255, тип **Word** – значения от 0 до 65535, тип **Longint** – значения от -2147483648 до +2147483647), вещественные числа – числа с десятичной точкой (тип **Real** – значения от $-2 \cdot 10^{39}$ до $+2 \cdot 10^{38}$) строковый и символьный типы (**String** и **Char**), а также логический тип **Boolean**. Используются также файловые переменные (тип **File**) для работы с файлами.



КАКИЕ ПРАВИЛА ИМЕНОВАНИЯ ПЕРЕМЕННЫХ В TURBO PASCAL?

Имя переменной в Turbo Pascal должно обязательно состоять только из одного слова (не допускается использование пробелов в имени) и содержать только латинские буквы (большие или маленькие – неважно, так как в Turbo Pascal они не различаются). В имени переменной можно использовать цифры и символ подчеркивания «_», но только не в начале. Имя переменной должно быть уникальным в пределах программы и его длина не должна превышать 63 символов.



ИЗ КАКИХ ОСНОВНЫХ ЧАСТЕЙ ДОЛЖНА СОСТОЯТЬ ПРОГРАММА НА TURBO PASCAL?

Программа на Turbo Pascal состоит из двух основных частей: описательной части и части, в которой реализована логика программы. В описательной части производится описание переменных, типов данных, констант и т.д., подключение модулей. Каждый раздел описательной части начинается со своего ключевого слова. Например, раздел описания переменных начинается со слова **Var**. В части логики программы пишутся непосредственно команды, операторы и т.д., задающие действия, производимые над описанными ранее переменными.

Необходимой частью любой программы на Turbo Pascal является часть логики программы, которая начинается со слова **Begin** и заканчивается словом **End** с точкой на конце.



КАКОВЫ ОСНОВНЫЕ ПРАВИЛА ПУНКТУАЦИИ ПРИ НАПИСАНИИ ПРОГРАММ НА TURBO PASCAL?

Основным пунктуационным правилом написания программ на Turbo Pascal является необходимость ставить знак «;» (точка с запятой) после каждой команды. Допускается использование пробелов между знаками препинания, операторами, командами, переменными и т.д.

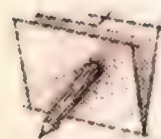


ЧТО ТАКОЕ ОПЕРАТОР ПРИСВАИВАНИЯ И ЗАЧЕМ ОН НУЖЕН?

Оператор присваивания предназначен для занесения в заранее описанную переменную какого-либо значения, не противоречащего типу переменной. Оператор присваивания состоит из знака двоеточия и знака «равно», имеет следующий вид:

Имя_переменной:= Значение_переменной

Значение переменной необязательно должно быть константой. Оно может представлять некоторое арифметическое или логическое выражение, результат которого и будет присвоен переменной.



КАКИЕ РЕЖИМЫ ОТОБРАЖЕНИЯ ИНФОРМАЦИИ ПРЕДУСМОТРЕНЫ В TURBO PASCAL?

В Turbo Pascal предусмотрено два режима отображения информации – текстовый и графический. Стандартным режимом является текстовый. Для перехода в графический режим используются специальные команды.

ГЛАВА 4.

ВЫВОД ИНФОРМАЦИИ НА ЭКРАН И ВВОД ДАННЫХ В ПРОГРАММУ



4.1. ВЫВОД ИНФОРМАЦИИ

Вывод информации на экран в текстовом режиме осуществляется процедурой `Write` (от англ. `Write` — писать), имеющей следующий вид:

`Write(<Выводимая информация>);`

Данная процедура выводит информацию на экран в символьном виде из текущего положения курсора слева направо. Если выводимая строка символов не помещается по ширине, то она будет перенесена на новую строку. Если при этом вывод осуществлялся в самую нижнюю строку, то все строки экрана будут сдвинуты вверх, в результате чего, самая верхняя строка пропадет, а внизу экрана появится пустая строка, в которую и будет продолжаться вывод (такой сдвиг строк называется скроллингом).

В качестве выводимой информации могут использоваться переменные или константы, предварительно описанные в соответствующих разделах. При этом переменные используются по имени, а константы либо по имени, либо задаются непосредственно значением.

`WriteLn(<Выводимая информация>);`

В отличие от процедуры `Write`, процедура `WriteLn`, после вывода информации, перемещает курсор в начало следующей строки. При этом информация выводится не с новой строки, а также как

и в случае `Write`, из положения курсора. Рассмотрим пример использования процедуры `WriteLn` для вывода информации с новой строки.

```
Program Vivod;  
Begin  
  WriteLn('Первая строка'); {вывод значения безыменной константы  
                             типа String со значением 'Первая строка'.  
                             Строка выводится из позиции каретки, за-  
                             тем каретка переводится в начало следу-  
                             ющей строки}  
  
  WriteLn('Вторая строка'); {вывод значения безыменной константы  
                             типа String со значением 'Вторая строка'.  
                             Строка выводится из позиции курсора, то  
                             есть с новой строки, затем курсор перево-  
                             дится в начало следующей строки}  
  
End.
```

В результате выполнения данной программы, надпись “Первая строка” выводится из положения курсора, а “Вторая строка” — с новой строки.

Если процедура `WriteLn` используется без параметров, то никакой информации на экран не выводится, а курсор перемещается на новую строку. Модифицируем предыдущий пример так, чтобы обе надписи выводились с новой строки. Для этого перед выводом первой строки нужно вызвать процедуру `WriteLn` без параметров.

```
Program Vivod;  
Begin  
  WriteLn; {вызов процедуры WriteLn без па-  
           раметров. Выводимая на экран ин-  
           формация отсутствует, курсор пе-  
           реводится в начало новой строки}  
  
  WriteLn('Первое предложение'); {выводится строка «Первое предло-  
                                   жение»}  
  
  WriteLn('Второе предложение'); {выводится строка «Второе предло-  
                                   жение»}  
  
End.
```

В случае необходимости, в процедурах `Write` и `WriteLn` можно указывать несколько параметров, разделенных запятыми:


```
Write(<Выводимая информация>, ..., <Выводимая информация>);  
WriteLn(<Выводимая информация>, ..., <Выводимая информация>);
```

Рассмотрим пример использования процедур вывода с несколькими параметрами.

```
program VivodSlogn;  
Const  
  Name = 'Владимир';  
Begin  
  WriteLn;  
  WriteLn('Меня зовут', Name);    {Будет выведено «Меня зовут Влади-  
                                  мир»}
```

4.2. УПРАВЛЕНИЕ РЕЖИМАМИ ВЫВОДА ИНФОРМАЦИИ

Основное требование для любого пользовательского интерфейса — удобство работы с ним. В понятие удобства работы входит цветное и шрифтовое оформление информации, представленной на экране, а также расположение различных частей информации относительно друг друга. При использовании текстового режима влиять на шрифтовое оформление невозможно, шрифт всегда является таким, каким его устанавливает операционная система или, например, программы русификации. Однако возможностей работы с цветом и расположением информации на экране вполне достаточно, чтобы пользователю было удобно работать с программой.

Процедуры, управляющие представлением информации на экране в текстовом режиме, собраны в стандартном (поставляемом с Turbo Pascal) модуле CRT. Для использования возможностей данного модуля его необходимо подключить к программе в разделе подключения модулей *Uses*:

```
Uses          {начало раздела подключения модулей}  
  CRT;        {подключение модуля CRT}  
Begin        {начало раздела описания логики программы}  
End.         {конец раздела описания логики программы}
```


Современные программы редко выводят на экран информацию в текстовом режиме. Текстовый режим, в основном используется для целей отладки и, соответственно, не предназначен для конечного пользователя. Поэтому, мы не будем рассматривать все возможности, предоставляемые модулем CRT, а ознакомимся только с некоторой их частью, необходимой для работы. Рассмотрим процедуры и функции управления выводом информации на экран в текстовом режиме, содержащиеся в модуле CRT, а также некоторые вспомогательные функции

ПРОЦЕДУРА CLRSCR

Название процедуры `ClrScr` является сокращением от `Clear Screen` (англ.) — очистить экран. Результатом выполнения данной процедуры является удаление всех символов, присутствующих на экране, и перемещение каретки в верхний левый угол экрана. Процедуру `ClrScr` рекомендуется вызывать в начале программы, чтобы информация, оставшаяся от других программ, не мешала пользователю, а также в конце программы, чтобы информация, выведенная нашей программой, не мешала работе с другими.

```
Program ClearScreen;  
Uses  
    CRT;  
Begin  
    ClrScr;  
    WriteLn('Да здравствует чистый экран!');  
    Write('Вся предыдущая информация была стерта');  
  
End.
```

ФУНКЦИЯ READKEY

Функция `ReadKey` (от англ. `Read Key` — читать клавишу) предлагает более развитые возможности, чем нужны нам в данный момент — она позволяет дождаться нажатия пользователем какой-либо клавиши и затем определить, какая клавиша была нажата. На основе данной функции обычно реализуется интерактивность

программ — возможность пользователя влиять на процесс работы программы. А пока мы будем использовать данную функцию для организации паузы в программе.

После передачи управления данной функции, программа приостанавливается до нажатия пользователем какой-либо символьной или управляющей клавиши (в список таких клавиш не входят клавиши «Shift», «Alt» и «Ctrl», «CapsLock», «NumLock» и «ScrollLock», а также «PrintScreen» и все специализированные клавиши, предназначенные для работы в Windows). Далее использование функции `ReadKey` пояснено на примере:

```
Program PrimerWaiting;  
Uses  
  CRT;  
Begin  
  ClrScr;  
  WriteLn('Вам нравится чистый экран?');  
  Write('Ничего не мешает работе');  
  Write(' с нашей программой!');  
  ReadKey;  
  ClrScr;  
End.
```

ПРОЦЕДУРЫ РАБОТЫ С ЦВЕТОМ

В любой момент времени выполнения программы, экран в текстовом режиме имеет две цветовых характеристики: текущий цвет выводимого текста и текущий фон выводимого текста. Эти характеристики показывают, какой цвет будут иметь символы, выводимые на экран процедурами `Write` и `WriteLn`, и на каком фоне они будут отображены.

Для изменения текущего цвета отображаемого текста используется процедура `TextColor` (от англ. **Text color** — цвет текста), имеющая следующий вид:

TextColor(<Номер цвета>);

В качестве Номера цвета используется число от 0 до 15, заданное переменной или константой (по имени или по значению).

Для всех возможных цветов в модуле CRT описаны константы. Рассмотрим цвета (см. табл. 4.1), которые можно использовать при выводе на экран, номера, которые им соответствуют, и константы, которые можно использовать вместо этих чисел.

Таблица 4.1. Цвета, используемые для вывода информации на экран и соответствующие им константы модуля CRT

Цвет	Номер	Константа модуля CRT
Черный	0	Black
Синий	1	Blue
Зеленый	2	Green
Морской волны	3	Cyan
Красный	4	Red
Фиолетовый	5	Magenta
Коричневый	6	Brown
Светло-серый	7	LightGray
Темно-серый	8	DarkGray
Светло-синий	9	LightBlue
Светло-зеленый	10	LightGreen
Светлый морской волны	11	LightCyan
Светло-красный	12	LightRed
Светло-фиолетовый	13	LightMagenta
Желтый	14	Yellow
Белый	15	White

Обратите внимание на то, что процедура TextColor устанавливает цвет только тех символов, которые будут выводиться после ее выполнения. Символы, уже находящиеся на экране, сохраняют свой цвет. В следующем примере, строка текста выводится красным цветом:

```
Program RedText;
Uses
  CRT;
Begin
  ClrScr;
  TextColor(4);
```

{подключение модуля CRT}
{начало раздела описания логики программы}
{очистка экрана}
{установка в качестве текущего цвета с номером 4 (красный). Начиная с этого места, вся информация будет выводиться красным цветом}


```
Write('Красный текст'); {вывод текстовой информации. Символы  
имеют красный цвет, так как именно он  
является текущим}  
  
ReadKey; {программа приостанавливается до нажатия  
пользователем клавиши. При этом вся  
информация, выведенная на экран видна}  
  
ClrScr; {очистка экрана}  
  
End.
```

В приведенном примере для установки текущего цвета текста используется номер цвета. Однако это не удобно, так как снижает читабельность программы, поэтому рекомендуется использовать вместо номеров цветов соответствующие им константы. Например, красному цвету соответствует константа `Red`.

Вторая цветовая характеристика выводимого на экран текста — текущий цвет фона. Для изменения текущего цвета фона используется процедура `TextBackGround` (от англ. **Text Background** — фон текста), имеющая следующий вид:

`TextBackGround(<Номер цвета>);`

Для цвета фона могут использоваться первые восемь цветов из шестнадцати доступных в текстовом режиме (см. табл. 4.1). Отметим, что цвет фона, также как и цвет текста, является текущей характеристикой, поэтому процедура `TextBackGround` влияет на цвет фона только тех символов, которые выводятся после ее вызова.

В следующем примере строка текста “Красное на черном” выводится красным цветом на черном фоне:

```
Program Hudoznik;  
Uses  
  
    CRT; {подключение модуля CRT}  
  
Begin {начало раздела описания логики  
      программы}  
  
    ClrScr; {очистка экрана}
```


<code>TextColor (Red) ;</code>	{установка в качестве текущего цвета, которому соответствует константа Red (красный). Начиная с этого места, вся информация будет выводиться красным цветом}
<code>TextBackGround (Black) ;</code>	{установка черного цвета в качестве текущего для фона (используется константа Black, соответствующая черному цвету). Начиная с этого места, вся информация будет выводиться на черном фоне}
<code>Write ('Красное на черном') ;</code>	{вывод текстовой информации. Символы имеют красный цвет, так как красный цвет является текущим, и выводятся на черном фоне}
<code>ReadKey;</code>	{программа приостанавливается до нажатия пользователем клавиши. При этом вся информация, выведенная на экран видна}
<code>ClrScr;</code>	{очистка экрана}
<code>End.</code>	

ПРОЦЕДУРЫ УПРАВЛЕНИЯ РАСПОЛОЖЕНИЕМ ИНФОРМАЦИИ

Мы уже умеем несколькими способами влиять на расположение текстовой информации: для того, чтобы строка текста выводилась в начале новой экранной строки, перед ее выводом следует использовать процедуру `WriteLn` (возможно, без параметров); а чтобы вывести строку не от начала экранной строки, а с некоторым отступом, можно перед ней напечатать с помощью процедуры `Write` необходимое количество пробелов. Однако данные методы не обеспечивают даже минимального удобства работы.

Приведем пример программы, выводящей строку “Строка, выровненная по центру” так, чтобы эта строка оказалась посередине экрана. Таким образом, данная строка должна находиться в экранной строке с номером 12 (всего на экране 24 строки,

следовательно, середина экрана $24 / 2 = 12$). По горизонтали первый символ выводимой строки должен находиться в позиции номер 25 (всего в экранной строке 80 символов, выводимая строка занимает 29 символов, следовательно, свободного места в экранной строке остается $80 - 29 = 51$ символов, слева и справа от выводимой строки свободного места должно быть поровну, то есть $51 / 2 = 25.5$, округляя до целого значения, получаем позицию 25).

```
Program CenterText;
Uses
  CRT;                                     {подключение модуля CRT}
Begin
  TextBackGround(Black);                 {черный цвет для фона}
  TextColor(White);                      {белый цвет для текста}
  ClrScr;                                {очистка экрана. Поскольку
                                         текущий цвет фона
                                         — черный, то весь экран
                                         будет окрашен этим
                                         цветом}

  WriteLn;                               {переводим строку 11}
  WriteLn;                               {раз, используя процедуру}
  WriteLn;                               {WriteLn без параметров}
  WriteLn;                               {соответствующее}
  WriteLn;                               {количество раз}
  WriteLn;
  WriteLn;
  WriteLn;
  WriteLn;
  WriteLn;                               {теперь строка 12 является
                                         текущей и информация
                                         будет выводиться в нее}

  Write('                                '); {выводим 24 пробела,
                                         соответственно, каретка
                                         перемещается в позицию
                                         25}
  Write('Строка, выровненная по центру'); {выводим строку}
```


ReadKey;	{программа приостанавливается до нажатия пользователем клавиши}
ClrScr;	{очистка экрана. Экран окрашивается в черный цвет. Все программы, выполняющиеся после завершения данной, будут использовать стандартную цветовую гамму}
End.	{конец раздела описания логики программы, информация, которая выводилась на экран в процессе работы программы, удалена}

Модуль CRT содержит более развитые средства для расположения текстовой информации на экране. Процедура `GotoXY` (от англ. **Go to X,Y** — переместить в точку с координатами координаты X, Y) позволяет переместить каретку в любую часть экрана. Данный

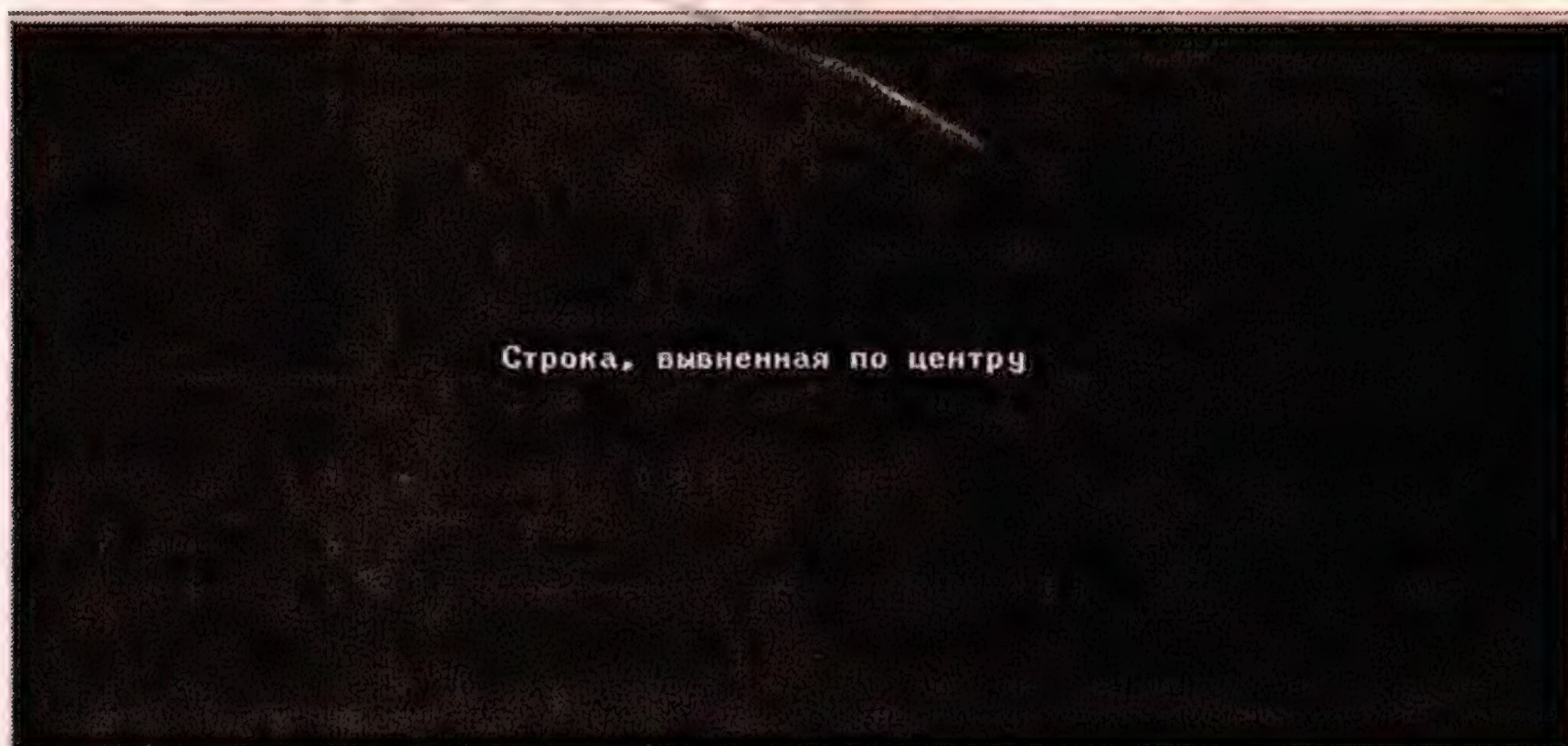


Рис. 4.1. Результат выполнения программы

оператор используется с двумя параметрами, определяющими новое положение каретки, и выглядит следующим образом:

GotoXY (<Горизонт. координата>, <Вертик. координата>);

Вертикальная координата определяет номер экранной строки, в которую должна переместиться каретка и может принимать значение от 0 (первая строка) до 23 (последняя строка), Горизонтальная координата определяет положение каретки в новой строке от левого края экрана и может изменяться от 0 (самое левое положение) до 79 (самое правое положение).

Таким образом, программу, приведенную в предыдущем примере можно реализовать следующим образом:

```
Program Center;
Uses
    CRT; {подключение модуля CRT}
Begin
    TextBackGround(Black); {черный цвет для фона}
    TextColor(White); {белый цвет для текста}
    ClrScr; {очистка экрана}
    GotoXY(24, 11); {перемещение каретки в строку 11 ( что отсчет
                    строк ведется от 0, поэтому это будет 12-ая
                    строка) и в символ номер 24 (соответсвенно,
                    это будет 25-ый символ в строке от левого
                    края экрана)}
    Write('Строка, выровненная по центру'); {вывод строки}

    ReadKey; {программа приостанавливается до нажатия
             пользователем клавиши}

    ClrScr; {очистка экрана}
End.
```

Иногда возникают ситуации, когда программе необходимо определить положение курсора. Это может понадобиться, например, при одновременном выводе информации и чтения строк, введенных пользователем, которые отображаются на экране автоматически (так называемый ввод с эхом). Возможности Turbo Pascal по вводу информации мы будем изучать позже, поэтому в данный момент рассмотрим простой пример на

определение положения каретки. Напишем программу, которая будет выводить на экран две строки: “Первая строка” и “Вторая строка”, причем вторая строка будет выводиться так, чтобы затереть последние два символа первой. Для решения данной задачи, необходимо вывести первую строку на экран процедурой `WriteLn`, определить координаты каретки, затем передвинуть ее в ту же экранную строку, но на два символа левее, то есть на предпоследний символ выведенной строки, и вывести вторую строку процедурой `Write`.

Для определения положения каретки в модуль `CRT` включены функции `WhereX` и `WhereY`. Данные функции служат для определения, соответственно, горизонтальной и вертикальной координаты текущего положения каретки. Отличия функций от процедур заслуживают особого внимания и обсуждаются в разделе, описывающем создание собственных процедур и функций (гл. 7), а пока, будем относиться к функциям, как к переменным, описанным в модуле `CRT`, и содержащим в каждый момент времени выполнения программы какие-либо значения.

Что за значение содержится в этой “переменной”, определяется назначением функции. Например, функция `ReadKey`, рассмотренная выше возвращает значение нажатой пользователем клавиши, а функция `WhereX` — горизонтальную координату каретки.

4.3. ЧТЕНИЕ ИНФОРМАЦИИ С КЛАВИАТУРЫ

Итак, пользователь может передать какую-либо информацию в программу, написанную на `Turbo Pascal`, всего двумя способами:

- путем ввода последовательности символов,
- путем ввода одиночных символов.

Ввод последовательности символов

Этот процесс выглядит следующим образом: программа вызывает процедуру чтения данных с клавиатуры и приостанавливает-

ся до тех пор, пока пользователь не нажмет на клавишу «Enter». Все это время пользователь может набирать на клавиатуре любые символы.

Сразу же после нажатия на какую-либо клавишу, символ, который ей соответствует, отображается на экране в позиции каретки (такой метод ввода имеет название ввод с эхом). Соответственно, использовать такой метод ввода информации с клавиатуры можно только в текстовом режиме, так как в графическом режиме, визуальные эффекты работы процедуры ввода непредсказуемы.

При наборе строки допустимо использование клавиши «BackSpace», стирающей символ, находящийся слева от каретки. После нажатия пользователем клавиши «Enter», введенная им строка попадает в некоторую переменную, описанную в программе, и программа может работать с ней в обычном порядке, то есть выводить на экран или использовать в различных выражениях.

Процедура ввода информации с клавиатуры называется Read (от англ. **Read** — читать), и имеет вид, напоминающий процедуру Write:

Read (<Приемник информации>) ;

В качестве параметра Приемник информации может использоваться переменная любого из числовых и символьных типов, например, Integer, Real, String или Char. В качестве данного параметра не могут использоваться переменные ссылочных типов, например, Pointer, и логические переменные типа Boolean, так как пользователь не может корректно ввести их значения, в связи с отсутствием у таких переменных визуального представления.

Также, в качестве данного параметра нельзя использовать константы, потому что значения констант не могут быть изменены в процессе выполнения программы. Это не касается типизированных констант, которые являются, по сути, переменными, с заранее заданными значениями.

Если необходимо последовательно ввести значения нескольких переменных, процедуру `Read` разрешается вызывать с любым количеством параметров:

`Read (<Приемник информации 1>, ..., <Приемник информации n>);`

В этом случае, после ввода значения очередной переменной, пользователь сразу же переходит к вводу значения следующей, и так далее, пока не будут заполнены значениями все переменные из заданного списка.

При вводе **числовых** данных таким методом, разрешается ввести значения переменных через пробел, не подтверждая каждый из них нажатием клавиши `Enter`, а подтвердить ввод можно после его полного завершения. Однако перед вызовом процедуры `Read`, рекомендуется вывести на экран строку, информирующую пользователя о том, какую именно информацию он должен ввести, поэтому использование процедуры `Read` с несколькими параметрами нежелательно, так как после ввода очередного значения, пользователь может не знать, что именно нужно вводить далее.

Рассмотрим пример работы с процедурой ввода `Read`. В качестве примера рассмотрим программу, которая запрашивает у пользователя его имя, сохраняет введенную пользователем последовательность символов в переменной, и, затем, выдает пользователю приветствие с использованием его имени

<code>Program YourName;</code>	
<code>Uses</code>	
<code> CRT;</code>	{подключение модуля CRT}
<code>Var</code>	
<code> Name: String;</code>	{описание строковой переменной для хранения имени пользователя}
<code>Begin</code>	
<code> ClrScr;</code>	{очистка экрана}
<code> Write('Введите Ваше имя: ');</code>	{отображаем приглашение}
<code> Read(Name);</code>	{получение строки, вводимой пользователем, и занесение ее в переменную Name}

WriteLn;	{переход на следующую строку}
Write('Привет, ', Name);	{последовательный вывод строки 'Привет, ' и имени пользователя, хранящегося в переменной Name}
ReadKey;	{ожидание нажатия клавиши}
End.	

При работе с процедурой `Read` появляется одна трудность — когда вводятся несколько текстовых строк подряд, все строки, кроме первой, не считываются. Это связано с тем, что первый вызов `Read` оставляет после себя, так называемый маркер конца строки, из-за которого все остальные вызовы `Read` считают, что пользователь нажимает клавишу `Enter`, вводя пустую строку. Для избежания такой ситуации, существует процедура `ReadLn`, не оставляющая маркера конца строки:

```
ReadLn (<Приемник информации 1>,
      ...,
      <Приемник информации n>);
```

Использование процедуры `ReadLn` аналогично использованию `Read`. Рассмотрим пример программы, запрашивающей у пользователя марку машины и модель, а, затем, выводящей сообщение с использованием введенной информации:

Program Car;	
Uses	
CRT;	{подключение модуля CRT}
Var	
CarBrand: String;	{описание строковой переменной для хранения марки машины}
CarModel: String;	{описание строковой переменной для хранения названия модели машины}
Begin	
ClrScr;	{очистка экрана}

<code>Write('Введите марку машины: ');</code>	{сообщение пользователю о необходимости ввести марку машины, переход на новую строку не производится, чтобы пользователь начал вводить свое имя в той же строке, где находится приглашение ко вводу}
<code>ReadLn(CarBrand);</code>	{получение строки, вводимой пользователем, и занесение ее в переменную CarBrand}
<code>WriteLn;</code>	{переход на новую строку}
<code>Write('Введите модель: ');</code>	{сообщение пользователю о необходимости ввести свою фамилию, переход на новую строку не производится, чтобы пользователь начал вводить свое имя в той же строке, где находится приглашение ко вводу}
<code>ReadLn(CarModel);</code>	{получение строки, вводимой пользователем, и занесение ее в переменную CarModel}
<code>WriteLn;</code>	{переход на новую строку}
<code>Write('Здравствуйте, у вас машина ', CarBrand, ' модель:', CarModel);</code>	
<code>ReadKey;</code>	{ожидание нажатия пользователем клавиши}
<code>End.</code>	

Ввод одиночного символа

Ввод одиночного символа производится уже знакомой функцией `ReadKey` модуля `CRT`:

`ReadKey;`

Функция `ReadKey` вызывается без параметров, ждет нажатия клавиши пользователем, и возвращает значение типа `Char`, со-

ответствующее нажатой клавиши. Рассмотрим пример работы с данной функцией.

Program ChtenieSimvola;	
Uses	
CRT;	{подключение модуля CRT}
Var	
Ch: Char;	{описание переменной для хранения символа, введенного пользователем}
Begin	
ClrScr;	{очистка экрана}
WriteLn('Нажмите клавишу!');	{сообщение пользователю о необходимости нажать клавишу}
Ch := ReadKey;	{ожидание нажатия пользователем клавиши и занесение символа, соответствующего нажатой клавише в переменную Ch}
Write('Вы нажали клавишу ', Ch);	
ReadKey;	{ожидание нажатия пользователем клавиши}
End.	

ШПАРГАЛКА. КОНТРОЛЬНЫЕ ВОПРОСЫ И ОТВЕТЫ ПО ТЕМЕ



КАКИЕ ОСНОВНЫЕ ПРОЦЕДУРЫ ПРЕДУСМОТРЕНЫ ДЛЯ ВЫВОДА ИНФОРМАЦИИ В ТЕКСТОВОМ РЕЖИМЕ?

Для вывода информации в текстовом режиме в Turbo Pascal предусмотрены две процедуры: **write** и **writeln**. Обе выводят на экран информацию, заключенную в скобках. При этом в скобках могут быть перечислены переменные, значения которых необходимо вывести на экран, а также константы, но они должны быть заключены в одинарные кавычки. Процедура **writeln** отличается от процедуры **write** тем, что после вывода осуществляет переход на следующую строку.



КАКАЯ ПРОЦЕДУРА ПОЗВОЛЯЕТ ВЫВЕСТИ ТЕКСТ В ТОЧНО ЗАДАННОЕ МЕСТО НА ЭКРАНЕ?

С помощью процедуры **GotoXY**(**<Гориз.координата>**,**<Вертик.координата>**) можно установить каретку (позицию, с которой будет осуществляться ввод/вывод) в нужное место на экране. Далее, используя стандартную процедуру **write**, можно осуществить вывод на экран нужной информации. При этом вывод будет осуществляться с установленной ранее позиции на экране.

Необходимо помнить, что положение по вертикали задается значением от 0 (первая строка) до 23 (последняя строка), а по горизонтали – от 0 (самое левое положение) до 79 (самое правое положение). Процедура **GotoXY** описана в модуле CRT, поэтому перед использованием **GotoXY** данный модуль должен быть подключен. Работает **GotoXY** только в текстовом режиме.



КАКИЕ ПРОЦЕДУРЫ ПОЗВОЛЯЮТ ЗАДАТЬ ЦВЕТОВОЕ ОФОРМ- ЛЕНИЕ ВЫВОДА (ЦВЕТ ФОНА И ЦВЕТ СИМВОЛОВ)?

В Turbo Pascal, а точнее в модуле CRT, предусмотрена процедура **TextColor** (**<Номер цвета>**), позволяющая установить цвет выводимого текста, а также процедура **TextBackGround** (**<Номер цвета>**), устанавливающая цвет фона, на котором будет выводиться текст. Обе процедуры работают только в текстовом режиме и влияют только на цвет и фон тех символов, которые выводятся после их вызова.



КАКИЕ ПРОЦЕДУРЫ ПРЕДУСМОТРЕНЫ ДЛЯ ПЕРЕДАЧИ В ПРОГРАММУ ДАННЫХ, ВВОДИМЫХ С КЛАВИАТУРЫ?

Для ввода информации с клавиатуры в Turbo Pascal предусмотрены две процедуры: **read** и **readln**. Вторая отличается от первой лишь тем, что переходит после своего выполнения на следующую строку. Формат процедур такой: **read(<Приемник информации>)** и **readln(<Приемник информации>)**. В качестве приемника информации, как правило, выступает переменная, в которую заносится вводимое с клавиатуры значение.

ГЛАВА 5.

ОПЕРАТОРЫ ЯЗЫКА



5.1. УСЛОВНЫЙ ОПЕРАТОР

Бывают конечно программы, которые выполняют заранее определенную, неизменную последовательность действий и на этом заканчивают свою работу. Однако гораздо чаще требуется писать программы, которые выполняют те или иные действия, в зависимости от тех или иных условий. Например, в зависимости от тех или иных действий пользователя, в зависимости от переданных программе данных и т.д. и т.п.

С этой целью в языке Turbo Pascal предусмотрен условный оператор `If...Then` (от англ. **If...Then** — Если...То), который позволяет выполнить какую-либо команду (процедуру, функцию, или оператор) в зависимости от истинности или ложности некоторого выражения. Выглядит условный оператор следующим образом:

If <Условие> Then <Команда>;

Здесь в качестве Условия может стоять логическое выражение любой сложности, в том числе содержащее переменные и константы, а также логические операторы `AND`, `OR` и `NOT`, рассмотренные выше.

После ключевого слова `Then` следует Команда, которая выполняется в случае истинности Условия. Обратите внимание, что команда всегда выполняется только одна. При этом на ее месте может стоять любая процедура или функция, а также какой-либо

оператор, в том числе и условный. Последний факт означает, что допускается использование вложенных условных операторов. Например:

```
If <Условие> Then  
    If <Условие> Then  
        If <Условие> Then <Команда>;
```

Кроме того, в Turbo Pascal предусмотрена еще расширенная версия условного оператора, предусматривающая указание команды, которую следует выполнить в случае ложности условия.

Расширенная версия условного оператора имеет следующий вид:

```
If <Условие> Then <Команда 1>  
    Else <Команда 2>;
```

Таким образом, в случае истинности условия выполняется Команда 1, а в случае его ложности выполнению подлежит Команда 2. Обратите внимание, что после Команды 1 точка с запятой не ставится! Если этим правилом пренебречь, компилятор впоследствии выдаст вам сообщение об ошибке и откажется компилировать программу.

В качестве примера использования условного оператора приведена программа, определяющая по двум углам является ли треугольник прямоугольным

```
program treugolnik;  
const  
    pi=180;  
var  
    a,b:real;  
    p,q:boolean;  
begin  
    writeln('Задайте два угла в треугольнике (в градусах):');  
    readln(a,b);  
    q:=false;  
    if ((pi-a-b)>0) then p:=true;  
    if ((pi-a-b)=pi/2) or (a=pi/2) or (b=pi/2) then  
        q:=true  
    else
```



```
If ((pi-a-b)<=0) then p:=false;  
if p then  
    writeln('Такой треугольник существует')  
else  
    writeln('Такой треугольник не существует');  
if p then  
    if q then  
        writeln('он прямоугольный')  
    else  
        writeln('но он не прямоугольный');  
end  
end.
```

5.2. СОСТАВНОЙ ОПЕРАТОР

Очень и очень часто в зависимости от истинности или ложности того или иного условия требуется выполнить не одно, а целый набор действий. Можно конечно под каждое действие писать свой условный оператор, но выглядит это не очень эффективно:

```
If a>b Then c:=10;  
If a>b Then d:=100-a;  
If a>b Then r:=c*d;  
If a>b Then write(r);
```

В Turbo Pascal предусмотрено использование так называемых составных операторов, которые представляют собой некую последовательность команд, заключенную между словами `Begin` и `end` (эти слова называются при этом операторными скобками). Это уже не те `Begin` и `end`, которые ограничивают раздел логики программы, это уже другие.

Принцип работы составного оператора заключается в том, что он воспринимается как один оператор, независимо от того, сколько других операторов заключено у него внутри. Это качество очень ценно, например, при использовании в условных и других операторах, которые предполагают в зависимости от какого-либо условия выполнение только одного оператора (команды). Если же в качестве команды использовать составной оператор, то таким

образом можно тот же условный оператор выполнять не одно, а целую последовательность действий.

Например, вышеприведенный пример, с использованием составного оператора можно записать следующим образом:

```
If a>b Then begin
    c:=10;
    d:=100-a;
    r:=c*d;
    write(r);
end
```

Следует обратить внимание, что ставить точку с запятой после оператора, предшествующего слову `end`, необязательно.

5.3. ОПЕРАТОР ВЫБОРА

Условный оператор по сути позволяет в зависимости от того или иного условия направить дальнейшее выполнение программы по одному из двух путей: либо по тому пути, который соответствует истинности условия, либо по тому пути, который соответствует ложности условия.

В последнем случае никаких операторов может не выполняться (если отсутствует часть оператора после/вместе со словом `else`), но отсутствие каких-то действий на определенном этапе — это тоже направление выполнения программы.

Тем не менее, бывает, что возникает необходимость реализовать выбор из большего числа возможных продолжений программы. Для этого в Turbo Pascal предусмотрен оператор множественного выбора `case`. Он имеет следующую структуру:

```
Case <Анализируемая переменная> Of

<Значение 1>: <Команда 1>;
<Значение 2>: <Команда 2>;
.....
<Значение N>: <Команда N>;
End;
```


Этот оператор позволяет в своем заголовке указать переменную, на основе различных значений которой должны производиться какие-либо действия, а, затем, описать сами действия (команды), привязав их к конкретным значениям переменной. В качестве команд как правило используются составные операторы.

Так же как и для условного оператора, для оператора выбора предусмотрен расширенный вариант, предусматривающий задание действия, которое должно произойти, если заданная переменная не принимает ни одного из перечисленных значений. Так же как и в условном операторе в операторе Case для этого используется секция Else:

```
Case <Анализируемая переменная> Of
  <Значение 1>: <Команда 1>;
  <Значение 2>: <Команда 2>;
  .....
  <Значение N>: <Команда N>;
Else <Команда N+1>
End;
```

При этом Команда N+1, находящаяся в секции Else в операторе Case, выполняется в случае, если Анализируемая переменная имеет значение, не равное Значениям от 1 до N.

В качестве примера приведена несложная программа, которая сначала запрашивает у пользователя два целых числа, а потом предлагает выбрать действие, которое над этими числами следует произвести. В зависимости от того, что пользователь выберет, то и будет сделано

```
program calc;
var x,y,n:integer;
begin
  write('Введите первое целое число x=');
  readln(x);
  write('Введите второе целое число y=');
  readln(y);
  writeln('Выберите действие, которое следует
предпринять');
```



```
writeln(' (введите соответствующий код) : ' );
writeln('          1-Сложение' ); \
writeln('          2-Вычитание' );
writeln('          3-Умножение' );
writeln('          4-Деление' );
readln(n);
case n of
    1:writeln('x+y=', x+y);
    2:writeln('x-y=', x-y);
    3:writeln('x*y=', x*y);
    4:writeln('x/y=', x/y)
end;
end.
```

5.4. ОПЕРАТОРЫ ЦИКЛА

В языке Turbo Pascal предусмотрено три оператора, позволяющих запрограммировать выполнение повторяющихся действий — повторяющиеся фрагменты программ. При этом повторное выполнение какого-либо фрагмента производится либо заданное количество раз (цикл `for`), либо до тех пор, пока остается истинным или ложным какое-либо заданное условие.

Данные операторы называются операторами цикла и будут рассмотрены далее в этом параграфе.

Цикл FOR

Оператор цикла со счетчиком циклов `For..To..Do` (от англ. **For..To..Do...** — От..До..Выполнить...) выполняет заданную Команду некоторое количество раз:

```
For <Параметр цикла> := <Нач. значение> To <Конеч. значение>
    Do <Тело цикла>;
```

В качестве Параметра цикла используется переменная любого целочисленного типа (`Integer`, `Byte`, `Word`), а в качестве Начального значения и Конечного значения — любые, допустимые для типа Параметра цикла значения. При этом в

качестве начального и конечного значений могут стоять арифметические выражения или другие переменные соответствующего типа

В качестве Тела цикла может использоваться любой ОДИН оператор – процедура, функция и другие операторы Turbo Pascal (в том числе условные, составные и операторы цикла).

В ходе выполнения цикла For в начале вычисляется Начальное значение, затем оно присваивается Параметру цикла. Далее идет циклическое повторение следующих действий:

1. Проверяется условие Параметр цикла $\leq$ Конечное значение. Если это условие не выполняется – оператор For заканчивает свою работу.
2. Выполняется Тело цикла
3. Производится увеличение значение параметра цикла на единицу и переход к п.1.

Необходимо иметь в виду, что условие Параметр цикла $\leq$ Конечное значение проверяется до выполнения Команды, которое может так и не состояться, если условие не выполняется с самого начала.

Кроме того, следует помнить, что приращение Параметра цикла всегда производится на одну и ту же величину (на +1). Однако существует другая форма оператора For:

```
For <Параметр цикла> := <Нач. значение> DownTo <Конеч. значение>  
Do <Тело цикла>;
```

При использовании цикла For с ключевым словом DownTo, а не To, на каждом шаге цикла происходит не увеличение Параметр цикла на единицу, а уменьшение на единицу, до Конечного значения.

Такая разновидность цикла For получила название цикла с обратным отсчетом.

ЦИКЛ WHILE

Для тех случаев, когда необходимо организовать повторяющееся действие до тех пор, пока выполняется какое-либо условие, в Turbo Pascal предусмотрен оператор цикла **While..do** (который так и называется «циклом пока»). Этот оператор имеет следующий формат:

```
While <Условие> Do <Тело цикла>;
```

Параметр Условие – выражение логического типа, Тело цикла – произвольный оператор. По ходу выполнения сначала проверяется истинно ли Условие, и если оно истинно, то выполняется Команда. Далее снова происходит проверка истинности условия и так дальше по кругу, пока условие не станет ложным.

В этой связи, чтобы не возникло бесконечного цикла (цикла, выполняемого бесконечное число раз и приводящего к зависанию программы), должно быть предусмотрены какие-либо изменения по ходу его выполнения, которые бы влияли на соблюдение условия и за какое-то конечное число повторений приводили к ложности условия цикла и к его завершению.

Обычно это реализуется таким образом, что в условии используется какая-либо переменная (или переменные) значение которых изменяется в теле цикла. При этом тело цикла зачастую представляет собой составной оператор.

ЦИКЛ REPEAT..UNTIL

В Turbo Pascal помимо цикла **while..do** предусмотрен еще один оператор цикла, повторение которого зависит от выполнения того или иного условия. Цикл этот – **Repeat..until** – имеет следующий вид:

```
Repeat  
  <Тело цикла>  
Until <Условие>;
```


Уже глядя на вид цикла можно увидеть, что в нем тело цикла гарантированно выполняется хотя бы один раз. Уже выполнив один раз тело цикла проверяется условие и в случае его ЛОЖНОСТИ (False) тело цикла будет выполнено еще раз и т.д. В этом второе отличие цикла Repeat..until от цикла while..do – цикл Repeat..until выполняется при ложности входящего в него условия.

Ну и, наконец, еще одной скорее особенностью чем отличием является то, что в теле цикла Repeat..until может присутствовать произвольное количество операторов. При этом их не надо заключать в операторные скобки (begin end), объединяя их в составной оператор. Служебными словами самого цикла достаточно четко и явно обозначаются границы тела цикла.

5.5. ОПЕРАТОР БЕЗУСЛОВНОГО ПЕРЕХОДА. МЕТКИ

В Turbo Pascal предусмотрен механизм так называемых меток. При этом в разделе Label в начале программы описываются переменные-метки:

```
Label
```

```
Metka01, pup, metka10
```

Далее, в произвольном месте раздела логики программы эта метка может быть установлена. При этом просто пишется имя метки с двоеточием на конце, а дальше идет дальнейший текст программы. При этом после двоеточия на конце метки точку с запятой уже ставить не надо.

Потом, с помощью специального оператора Goto можно из любого места программы, где используется данный оператор, перейти к тому месту, где установлена метка. При этом в ходе выполнения программа выполняется последовательно до тех пор, пока не встретит оператор Goto, повстречав который программа перескочит сразу к тому месту своего кода, где установлена метка и далее будет выполняться уже с этого места.

Формат использования оператора Goto следующий:

Goto <метка>;

Общий формат использования меток выглядит следующим образом:

Label

Metka01, pup, metka10

.....

Begin

.....

pup:

.....

goto metka01;

.....

Metka01:

.....

goto pup;

.....

Использование меток нежелательно и является плохим тоном программирования, так как программа становится запутанной и логически нестройной. Рекомендуется обходиться другими средствами языка (условными операторами, циклами) и продумывать структуру программы. В 99 случаях из 100 можно обойтись без использования меток, причем программа в итоге будет выглядеть лучше и понятнее.

ШПАРГАЛКА. КОНТРОЛЬНЫЕ ВОПРОСЫ И ОТВЕТЫ ПО ТЕМЕ



ЧТО ТАКОЕ УСЛОВНЫЙ ОПЕРАТОР?

Условный оператор – это оператор, который позволяет организовать в программе выполнение тех или иных действий в зависимости от истинности или ложности какого-либо условия. Различают обычную

(If <условие> then <команда в случае истинности условия>) и расширенную форму условного оператора (If <условие> then <команда в случае истинности условия> else <команда в случае ложности условия>).



Что такое цикл и какие циклы предусмотрены в Turbo Pascal?

Под циклом в языках программирования понимается оператор, который позволяет организовать повторяющееся выполнение заданных действий определенное количество раз или в соответствии с каким-либо условием. В Turbo Pascal предусмотрено три цикла: цикл со счетчиком (цикл **For**), цикл ПОКА или цикл с предусловием (цикл **While..Do**) и цикл с постусловием (цикл **Repeat..Until**)



Что такое цикл со счетчиком (цикл For)?

Цикл **For** (его еще называют циклом со счетчиком) предназначен для организации выполнения определенных действий заданное количество раз. В качестве счетчика выступает переменная, значение которой увеличивается или уменьшается на 1 при каждом повторном выполнении указанных действий (тела цикла). По достижении значения счетчика заданной величины выполнение цикла завершается.



Что такое цикл ПОКА (цикл While)?

Цикл **While** (его обычно называют циклом ПОКА или циклом с постусловием) предназначен для выполнения заданных действий (тела цикла), пока остается ложным какое-либо условие. Кстати говоря, именно поэтому его и называют часто циклом ПОКА.

ГЛАВА 6.

СЛОЖНЫЕ ТИПЫ ДАННЫХ



6.1. МАССИВЫ

Ранее рассмотренные типы данных предназначены для использования в программе одиночных объектов и оперирования ими (числами, символами, строками и т.п.). Однако в языке Turbo Pascal предусмотрено также использование объектов, являющихся по своей сути объединениями нескольких элементов.

Так, массив – это формальное объединение нескольких однотипных объектов, рассматриваемое как единое целое. Важность наличия массивов в Turbo Pascal трудно переоценить, так как именно через массивы очень часто осуществляется обработка относительно больших объемов данных, а именно для таких целей (обработки/анализу более-менее больших объемов данных) во многих случаях как раз и пишутся программы.

В виде массива очень удобно представить, например перечень значений какой-либо величины или результаты каких-либо измерений.

Доступ к отдельным элементам массива осуществляется по их индексу. В этом смысле можно дать еще одно определение, отражающее механизм работы с массивами: **массив** – это группа переменных одного типа (**элементов массива**), названная одним именем и имеющая индексный механизм доступа к каждой переменной этой группы.

Описание массива включает в себя указание из переменных какого типа должен состоять массив, сколько в нем должно быть элементов, и какие индексы должны быть использованы для доступа к его элементам:

Var

```
<Имя массива>: Array [<Нижний индекс>..<Верхний индекс>]  
Of <Тип элементов>;
```

Для Имени массива применяются идентификаторы, отвечающие тем же правилам, что и имена переменных и других элементов программы.

Параметры массива Нижний индекс и Верхний индекс должны являться константами и определяют пределы изменения индекса и, соответственно количество элементов, которые содержатся в массиве. Например, если в качестве Нижнего индекса использовано значение 5, а в качестве Верхнего индекса — 7, то массив будет включать в себя три элемента, доступ к которым может быть осуществлен по индексам 5, 6 и 7.

В качестве Типа элемента может быть использован любой необходимый для решения поставленной задачи тип данных, например Integer или String, в том числе и другой массив.

Необходимо иметь в виду, что объем оперативной памяти, занимаемой переменными, описываемыми в разделе описания переменных Var не должен превышать 64-х килобайт. Таким образом существует ограничение на максимальное количество элементов заданного типа в массиве. Для массива элементов типа Integer, например, такое ограничение — 32408.

type

```
massiv = array [0..20] of char;  
kukan = array [byte] of integer;
```

var

```
a: massiv;  
b, sum: kukan;  
i: byte;
```

begin


```
.....  
sum[15]:= b[i+1]*2; {иллюстрация обращения к отдельному}  
                   {элементу массива по его индексу}  
                   {При этом в качестве индекса может  
                   выступать переменная}
```

Определить переменную как тот или иной массив можно и непосредственно в разделе Var без предварительного описания типа массива:

```
var  
    a, hoh : array [0..20] of char;
```

В качестве типа массива может в свою очередь выступать также массив. Таким образом получается массив массивов – двумерный массив. По аналогии могут быть трехмерные массивы (массивы из массивов массивов), четырехмерные массивы и т.д.

Вот описание двумерного массива:

```
type  
    massiv = array [0..20] of array [-5..5] of real;
```

Кстати говоря, оно может быть сведено к более краткой форме:

```
type  
    massiv = array [0..20, -5..5] of real;
```

Трехмерный массив будет выглядеть примерно так:

```
type  
    massiv = array [0..20, -5..5, a..g] of real;
```

Если при обращении к элементу обычного одномерного массива используется один индекс, указываемый в квадратных скобках, то при обращении к элементу двумерного массива должны указываться два индекса через запятую, например a[10,5]. Соответственно для обращения к элементу трехмерного массива используются три индекса и т.д.

В Turbo Pascal предусмотрена интересная возможность одним оператором присваивания передать все элементы одного массива другому массиву того же типа. Делается это следующим образом:

```
type
  massiv = array [0..20] of char;
  kukan = array [byte] of integer;
var
  a: massiv;
  b, sum: kukan;
  i: byte;
begin
  .....
  b := sum;
```

Последовательный перебор элементов массива как правило осуществляется с помощью цикла `for`. При этом для двумерных массивов (их еще называют матрицами) используется два цикла `for`, вложенных один в другой.

В качестве примера работы с массивом, ввода и перебора его элементов, рассмотрим программу, которая в квадратной матрице (двумерном массиве) меняет местами строку с максимальным элементом на главной диагонали со строкой с заданным номером `m`. Индексы элементов, расположенных на главной диагонали равны, например `a[1,1]`, `a[2,2]` и т.д.

```
program Massiv;
var
  a: array[1..30,1..30] of integer;
  j, n, k, m, max, at: integer;
begin
  writeln('Введите размерность матрицы:');
  readln(n);
  write('Введите номер строки m=');
  readln(m);
  for i:=1 to n do
    for j:=1 to n do
      begin write('a[' , i, j, ']=');
        readln(a[i,j])
      end;
```



```
writeln('Ваша матрица:');
for i:=1 to n do
  begin for j:=1 to n do
        write(a[i,j]:4);
        writeln
      end;
max:=1;
for i:=1 to n do
  if a[max,max]<a[i,i]
    then max:=i;
for i:=1 to n do
  begin at:=a[m,i];
        a[m,i]:=a[max,i];
        a[max,i]:=at
  end;
writeln('Полученная матрица:');
for i:=1 to n do
  begin for j:=1 to n do
        write(a[i,j]:4);
        writeln
      end
end.
end.
```

6.2. СТРОКИ

ОПИСАНИЕ И ИСПОЛЬЗОВАНИЕ СТРОК

Поскольку работа с текстовыми строками довольно частая вещь в практике программирования вообще и на Turbo Pascal в частности, в Turbo Pascal предусмотрен специальный тип данных для этого – строковый тип `string` (в переводе с английского – строка).

Тип `string` во многом напоминает одномерный массив символов `array[0..N] of char`, но, в отличие от последнего, количество символов в строке может меняться по ходу выполнения программы от 0 до N. Где N – это максимально допустимо количество символов в строке, указываемое при описании строковой переменной. Это значение может быть больше 255.

Описание строковых переменных выглядит следующим образом:

```
var  
  s : string [30];
```

В Turbo Pascal разрешается не указывать максимальное допустимое количество символов в строке. В этом случае за максимум принимается значение 255:

```
var  
  s : string;
```

Обращение к элементам строки (символам в строке) осуществляется по индексу – порядковому номеру символа в строке. При этом необходимо знать, что нумерация символов начинается не с 1, а с 0.

```
var  
  s : string [30];  
begin  
  s := 'Слѣва туду'  
  s[3] := 'a'
```

ПРОЦЕДУРЫ И ФУНКЦИИ, ПРЕДУСМОТРЕННЫЕ В TURBO PASCAL ДЛЯ РАБОТЫ СО СТРОКАМИ

В Turbo Pascal существует возможность складывать строки, то есть сцеплять их с помощью стандартной операции сложения «+». Например:

```
s1 := s2 + s3;
```

При этом вторая строка дописывается в конец первой. Если размер результирующей строки станет превышать установленное для нее максимальное значение, то все «лишние» символы будут отброшены.

Остальные операции над строками осуществляются с помощью перечисленных далее стандартных процедур и функций:

- **concat (s1, s2, ..., sN)** – функция типа `String`, в качестве результата выдающая строку, являющуюся сцеплением перечисленных в скобках строк. Например, вышеприведенную операцию сложения двух строк можно записать, как `s1 := concat(s2, s3)`. По названию этой функции операция сцепления(сложения) строк получила название **конкатенация**.
- **delete (st, index, count)** – эта процедура из строки `st` начиная с позиции `index` удаляет `count` символов.
- **insert (subst, st, index)** – процедура, вставляющая в строку `st` другую строку `subst` начиная с позиции `index` (в строке `st`).
- **copy (st, index, count)** – эта функция выдает строку, взятую из строки `st` начиная с позиции `index`. Количество взятых символов в этой строке равно `count`.
- **length (st)** – данная функция выдает значение типа `integer`, являющееся текущей длиной строки `st` в символах. Таким образом, чтобы присвоит какой-либо переменной текущий размер строки `s` нужно выполнить следующее `p := length(s)`.
- **pos (subst, st)** – функция, разыскивающая вхождение подстроки `subst` в строку `st`. В качестве результата своей работы она выдает либо 0 – если вхождения обнаружено не было, либо целое положительное число (`integer`), являющееся номером позиции, с которой выявлено вхождение. При этом оценивается только первое вхождение. Если вхождений несколько, то обнаружено будет только первое из них и только его позиция будет выдана в качестве результат функции.
- **Val (st, x, code)** – если переменная или константа типа `String` содержит символьное представление вещественного или целого числа, например `'12.8'` или `'124'`, то с помощью процедуры `Val` можно преобразовать это

число в значение переменной типа `Real`. Здесь `st` – это сама исходная строка, `x` – это переменная, которой будет передано получившееся вещественное число. Целочисленный параметр `code` содержит 0, если преобразование произошло успешно, в противном случае – номер места в строке, в котором произошла ошибка.

- **Str (x, st)** – процедура, обратная предыдущей, – преобразует число из переменной `x` в строку, которая присваивается переменной `st`.

В качестве примера рассмотрим программу, которая считает, сколько раз встречается указанное пользователем слово в строке

```
program strokaincl;
var
  s,s1,s2:string;
  k,i:integer;
begin
  write('Введите исходную строку:');
  readln(s);
  write('Введите слово, которое следует искать: ');
  readln(s1);
  s2:=s;      {сохраняем исходную строку во второй
               переменной, чтобы не утратить}
  k:=0;      {счетчик вхождения искомого слова в строку}
  while pos(s1,s)>0 do
    begin k:=k+1;
           delete(s,pos(s1,s),length(s1));
    end;
  writeln(k);
end.
```

И еще один пример программы, которая подсчитывает в строке количество букв А и В. Если букв А оказывается больше, чем букв В, то программа стирает из строки все буквы В и выводит строку на экран.


```
program stroka2;  
var  
    s:string;  
    k,l,i:integer;  
begin  
    write('Введите исходную строку: ');  
    readln(s);  
    k:=0;           {счетчик букв А в строке}  
    l:=0;           {счетчик букв В в строке}  
    for i:=1 to length(s) do  
        begin if s[i]='A' then k:=k+1;  
               if s[i]='B' then l:=l+1  
        end;  
    writeln('Количество букв А=',k);  
    writeln('Количество букв В=',l);  
    if k>l then  
        begin for i:=1 to length(s) do  
                if s[i]='B' then begin delete(s,i,1);  
                                     i:=i-1  
                                end;  
        end;  
    write(s)  
end.
```

СРАВНЕНИЕ СТРОК

Для строковых переменных (переменных типа `String`) в Turbo Pascal предусмотрена операция сравнения, то есть для двух строк можно определить, какая из них “больше” или “меньше” другой.

Сравнение строк основано на кодах символов, из которых состоят строки, и длине строк. Например, строка “А” меньше строки “Z”, так как символ “А” имеет код меньший, чем у символа “Z”.

Вообще все символы-буквы латиницы и кириллицы расположены в таблице символов ASCII по алфавиту, что позволяет применять сравнение строк, например, при упорядочивании массива: если упорядочивать массив элементов типа `String` по возрастанию, то элементы (строки) расположатся в алфавитном порядке. Что

касается всех остальных символов, которые можно выводить на экран, то они разбросаны по таблице символов (см. табл. 6.1)

Таблица 6.1. Коды некоторых видимых символов таблицы ASCII

Код	Символ
33-47	! " # \$ % & ' () * + , - . /
48-57	0 - 9
58-64	: ; < = > ? @
65-90	A - Z
91-96	[\] ^ _ `
97-122	a - z
123-126	{ } ~
128-159	А - Я
160-175	а - п
224-239	р - я

Если какие-либо две строки идентичны по набору символов, но одна из них имеет длину, большую, чем другая, то она считается большей. Например, строка “ABCD” больше чем строка “ABC”. Заметим, что длина строки — всего лишь дополнительный критерий, и, например, строка “AAAA” будет меньше строки “BBB”, хотя и имеет большую длину.

6.3. ЗАПИСИ

ОПИСАНИЕ ЗАПИСЕЙ

Записи представляют собой объединения нескольких элементов, которые, в отличие от элементов массива, могут быть разнотипными. Элементы записи называются полями, каждое поле записи имеет свое имя, уникальное в пределах записи.

Запись является типом данных и описывается в разделе описания типов данных Type. После описания типа данных, его можно использовать для создания переменных этого типа в обычном порядке. Описание записей как типов данных имеет следующий вид:

Type	
<Имя типа> = Record	{заголовок описания записи}
<Имя переменной 1>: <Тип>;	{описание "поля" записи}
.....	
<Имя переменной n>: <Тип>;	{описание "поля" записи}
end;	{окончание описания типа}

Рассмотрим пример описания нового типа данных — записи, содержащей информацию о сотруднике.

Program Person;	
Type	{начало раздела описания типов данных}
Person = Record	{заголовок записи}
Name: String;	{поле, представляющее имя сотрудника}
Age: Integer;	{поле, представляющее возраст сотрудника}
Address: String;	{поле, представляющее адрес сотрудника}
end;	{окончание описания записи}
.....	
Begin	{организация работы со структурами данных}
.....	
End.	

Описанная таким образом запись с именем Person является типом данных, а не переменной, то есть не существует на самом деле, поэтому к ней нельзя обращаться для занесения и получения информации.

Использовать запись можно только для описания переменной этого типа. Аналогичным свойством обладают все типы данных, которые мы используем. Например, Integer — является типом данных и не может быть использован в строке вида Integer := 10.

Описание переменных-записей, или, как их принято называть **экземпляров записей**, осуществляется по обычным правилам в разделе описания переменных Var:


```
Program Person2;  
Type  
    Person = Record {описание записи, как типа}  
        Name: String;  
        Age: Integer;  
        Address: String;  
    end;  
Var  
    Secretar: Person; {описание экземпляра записи типа  
                        Person с именем Secretar}  
    .....  
Begin {организация работы со структурами  
    ..... данных}  
End.
```

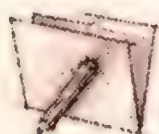
ОПЕРАЦИИ НАД ЗАПИСЯМИ И ПОЛЯМИ ЗАПИСЕЙ

Для обращения к какому либо полю некоторой записи используется имя этой записи и имя поля в его составе, разделенные точкой:

<Имя экземпляра записи>.<Имя поля>

С полями экземпляров записей можно производить любые операции, допустимые для типа данных этого поля, то есть указывать в качестве параметров процедур и функций, использовать как операнды для арифметических и логических операторов, и так далее.

ШПАРГАЛКА. ПОЛЕЗНЫЕ МЕТОДЫ, ПРИЕМЫ И ПРАВИЛА



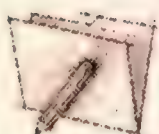
Что такое массив?

Массив – это совокупность однотипных данных, хранящихся в последовательных ячейках памяти и имеющих одно общее имя. Составляющие массива – ячейки – называются элементами массива. Все они пронумерованы по порядку, а сам номер называется индексом элемента массива. Элементы одномерных массивов имеют один индекс, двумерных – два индекса и т.д. Обращение к какому-либо элементу массива осуществляется через имя массива со стоящим на конце индексом элемента, заключенным в квадратные скобки. Например, $a[3]$ или $d[5, 1]$.



Метод пузырька – стандартный метод реструктуризации элементов массива

Метод пузырька – это наиболее распространенный метод реструктуризации элементов массива (или вообще группы каких-либо данных), когда требуется переупорядочить элементы массива в соответствии с каким-либо правилом (например, по убыванию или возрастанию). Принцип заключается в том, что чтобы поменять местами два значения в массиве, нужно сначала значение второго элемента поместить в какую-либо вспомогательную переменную (выполняющую роль пузырька), затем поместить во второй элемент массива значение первого элемента, а затем в первый элемент – значение второго, взятое из вспомогательной переменной.



Наличие цикла в программе с массивами

Если в программе присутствует массив, то в ней обязательно должен быть хотя бы один оператор цикла. Если в программе с массивом оператор цикла отсутствует, то программа написана неправильно.



Что такое запись?

Запись – это структура данных, представляющая собой совокупность нескольких разнотипных данных, объединенных под одним именем. Эти данные называются полями записи и имеют свои имена внутри записи. Обращение к полю записи осуществляется путем указания имени записи и имени поля, разделенных точкой

ГЛАВА 7.

ПРОЦЕДУРЫ И ФУНКЦИИ



Довольно часто, особенно при разработке больших программ, возникает необходимость одну и ту же последовательность действий запрограммировать несколько раз. Например, если программе вычисляются корни нескольких квадратных уравнений, то для каждого из уравнений придется писать одну и ту же последовательность действий по вычислению корней.

Даже на первый взгляд это кажется неэффективным, а если подобных повторяющихся фрагментов в программе становится очень много, да еще если они содержат не две-три программных строки, а десятки строк, то программа, во-первых, приобретает очень громоздкий вид, а во-вторых, в ней становится очень сложно разбираться даже человеку, написавшему ее. Кроме того, если впоследствии понадобится внести какие-либо изменения в повторяющийся код, то придется это делать во всех местах, где он встречается. Отладка программы в результате приобретает абсолютно хаотичный характер: отслеживать исправления становится сложно, проверять где внесены исправления, а где — тоже сложно и т.д. и т.п.

Однако в языке Turbo Pascal предусмотрены средства, позволяющие разрешить данную проблему путем использования так называемых подпрограмм. При этом повторяющийся фрагмент команды записывается в одном месте, получает свое имя и потом по этому имени может быть вызван из любого места программы. В случае необходимости внесения поправок они вносятся только в

одном месте — где присутствует код подпрограммы. В местах вызова (выполнения) этого кода уже никаких поправок вносить не нужно.

В Turbo Pascal возможно использование двух видов подпрограмм — процедур и функций. Оба вида будут рассмотрены далее в этой главе и позволяют существенно оптимизировать разработку программы, сделать ее структуру более понятной.

7.1. ПРОЦЕДУРЫ

Под процедурой в Turbo Pascal понимается логически законченная часть программы, описываемая отдельно от раздела описания логики программы и имеющая уникальное в пределах программы имя, по которому она может быть вызвана.

Структура процедур в принципе повторяет структуру обычных программ — в ней также присутствуют разделы описания переменных, констант, типов, раздел логики и т.д. Единственное отличие заключается в том, что по своей структуре процедура, в отличие от программы, не может иметь раздела подключения модулей. В остальном же они практически идентичны.

Procedure <Имя>;	Заголовок процедуры
Label	Раздел описания меток безусловного перехода
Const	Раздел описания констант
Type	Раздел описания типов данных
Var	Раздел описания переменных
Procedure	Раздел описания подпрограмм — процедур и функций, содержащих некоторые элементы логики процедуры
Function	
begin	Начало раздела описания логики процедуры
end;	Раздел описания логики процедуры Конец раздела описания логики процедуры

Назначение разделов процедуры аналогично назначению таких же разделов программы, при этом метки безусловного перехода, константы, типы данных, переменные, процедуры и функции, описанные в процедуре являются локальными, то есть доступны только из процедуры, в которой они описаны. При этом все перечисленные элементы программы, доступны в процедуре точно так же, как и в разделе описания логики самой программы.

Если в процедуре и программе описана переменная с одним и тем же именем, что допустимо, то при обращении к данному имени из раздела описания логики программы будет использована переменная, описанная в разделе описания переменных программы, а при обращении к этому же имени из раздела описания логики процедуры будет использована переменная, описанная в разделе описания логики процедуры.

Структуры, описанные в разделах описания программы, а не процедур и функций называются **глобальными**.

7.2. ФУНКЦИИ

Функции — это процедуры, которые возвращают значения, что позволяет использовать их в арифметических или логических выражениях также как переменные или константы. Структура функции аналогична структуре процедуры, за исключением заголовка, в котором указывается, значение какого типа возвращает функция. Также, вместо ключевого слова Procedure используется ключевое слово Function:

```
Function <Имя функции>: <Тип возвращаемого значения>;
```

Для указания компилятору, какое именно значение должна возвращать функция в разделе описания логики функции должно использоваться присвоение следующего вида:

```
<Имя функции> := <Значение>;
```


Таким образом, описание функции, возвращающей значение дискриминанта для нашей программы, может выглядеть

7.3. ПРОЦЕДУРЫ И ФУНКЦИИ С ПАРАМЕТРАМИ

Во всех языках программирования для передачи какой-либо информации в процедуры и функции используется механизм параметров — описание структуры данных, необходимой процедуре или функции для работы, заполняемой при вызове подпрограммы. Описание такой структуры данных происходит в заголовках процедур и функций после имени и имеет следующий вид:

```
Procedure <Имя> (<Параметр 1>: <Тип параметра 1>; ...;  
                <Параметр N>: <Тип параметра N>);
```

или для функций:

```
Function <Имя> (<Параметр 1>: <Тип параметра 1>; ...;  
              <Параметр N>: <Тип параметра N>): <Тип значения>;
```

Процедуры и функции обращаются к описанным в их заголовке параметрам, заполняемым вызывающей подпрограммой или программой, по имени, также как и к локальным переменным, но только для чтения. Таким образом, параметру нельзя присвоить новое значение, а можно только прочесть текущее.

При вызове параметры могут задаваться переменными или константами соответствующих типов, перечисленными в последовательности, заданной при описании подпрограммы.

Использование параметров в подпрограммах аналогично использованию локальных переменных, за исключением невозможности изменить их значения. В остальном же, с ними можно делать все то же самое, что и с переменными, например, передавать в качестве параметров другим процедурам и функциям.

7.4. ПЕРЕДАЧА ПАРАМЕТРОВ ПО ЗНАЧЕНИЮ И ПО ССЫЛКЕ. ПРОЦЕДУРЫ И ФУНКЦИИ С ПАРАМЕТРАМИ-ПЕРЕМЕННЫМИ

Функции очень удобны в использовании, однако имеют довольно существенное ограничение — функция может возвращать только одно значение, хотя можно привести множество примеров, когда на основе нескольких параметров рассчитываются несколько других. Соответственно, появляется проблема, как вернуть несколько каких-либо значений вызывающей подпрограмме или программе. Для решения проблемы механизм параметров подпрограмм дополнен, так называемыми, параметрами-переменными.

Перед описанием таких параметров в заголовках подпрограмм указывается ключевое слово `Var`, а при вызове подпрограмм, в которых параметры описаны таким образом, в качестве этих параметров должны использоваться переменные, а использование констант недопустимо. Это связано с тем, что подпрограмма, имеющая параметр-переменную может изменять ее значение, причем, на этапе разработки программы может быть неизвестно, какая именно переменная будет использоваться в качестве параметра, но значение, присвоенное параметру-переменной, в качестве которого она задана, попадет именно в нее.

На низком уровне такая возможность реализована разными методами передачи параметров в подпрограммы. Простые параметры, рассмотренные нами выше, **передаются по значению**, поэтому все, что может знать подпрограмма о переданном ей параметре — это его значение. Тогда как параметры-переменные **передаются по ссылке**, то есть подпрограмме передается не значение параметра, а адрес переменной, которая в качестве параметра задана. Соответственно, подпрограмма может занести в переданный ей адрес любую информацию допустимого типа.

Несмотря на такие серьезные внутренние отличия, на уровне языка Turbo Pascal нет никакой разницы в работе с простыми параметрами и параметрами-переменными, за исключением того, что параметры-переменные могут использоваться в левых частях

операторов присваивания, то есть их значения могут быть изменены.

7.5. ПРИМЕР ИСПОЛЬЗОВАНИЯ

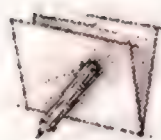
В качестве практического примера использования функций рассмотрим программу, вычисляющую определитель матрицы:

```
Program Determinant;
const
    N1=10;
type
    Matrice=array[1..N1,1..N1] of real;
var
    A:matrice;
    I,J,N:integer;
    D:real;
Function Det(A:Matrice;N:integer):real;
var
    B:matrice;
    I:integer;
    T,Mn,S:real;
Function Minor(var C:matrice;A:Matrice;N,I,J:integer):real;
var
    Im,Jm,Ia,Ja,Nm:integer;
begin
    Nm:=N-1; Im:=1; Ia:=1;
    while Im<=Nm do
        if Ia<>I then
            begin
                Jm:=1; Ja:=1;
                while Jm<=Nm do
                    if Ja<>J then
                        begin
                            C[Im,Jm]:=A[Ia,Ja];
                            Ja:=Ja+1; Jm:=Jm+1;
                        end
                    else Ja:=Ja+1;
                end
            end
        end
    end
```



```
        Ia:=Ia+1; Im:=Im+1;
    end
    else Ia:=Ia+1;
end; (*Minor*)
begin
    if N=1 then Det:=A[N,N];
    if N=2 then Det:=A[1,1]*A[2,2]-A[2,1]*A[1,2];
    if N>2 then
        begin
            S:=0;
            for I:=1 to N do
                begin
                    Mn:=Minor(B,A,N,I,1);
                    if (I mod 2)=1 then begin
                        T:=Det(B,N-1);
                        S:=S+T*A[I,1];
                    end
                    else begin
                        T:=Det(B,N-1);
                        S:=S-T*A[I,1];
                    end
                end
            end;
            Det:=S;
        end;
end; (*Determ*)
begin
    Write('Введите порядок матрицы N: '); readln(N);
    for I:=1 to N do
        begin
            writeln('Введите элемент строки ',I:2);
            for J:=1 to N do readln(A[I,J]);
        end;
    D:=Det(A,N);
    Writeln('Определитель равен ',D:7:4);
    readln;
end.
```


ШПАРГАЛКА. КОНТРОЛЬНЫЕ ВОПРОСЫ И ОТВЕТЫ ПО ТЕМЕ



Что такое подпрограмма?

Подпрограммой называется именованная логически законченная группа операторов, которую можно вызвать для выполнения по имени любое количество раз из различных мест программы. В Turbo Pascal для организации подпрограмм предусмотрены процедуры и функции.



Что такое процедура и функция? Чем они отличаются?

Процедура – это независимая поименованная часть программы, предназначенная для выполнения определенных действий. Она состоит из заголовка и тела. Структура процедуры практически такая же, как и структура обычной программы.

После однократного описания процедуру можно вызвать по имени из последующих частей программы. Когда процедура выполнит свою задачу, программа продолжится с оператора, следующего непосредственно за оператором вызова процедуры. Имя процедуры не может использоваться в арифметических и логических выражениях, то есть процедура не может быть вызвана оттуда.

Функция аналогична процедуре, но имеет два отличия. Во-первых, функция передает в точку вызова определенное значение, а во-вторых, имя функции может входить в арифметические и логические выражения в качестве операнда.



Что такое глобальные и локальные переменные?

Переменные, описанные в программе, могут использоваться в подпрограммах – они являются глобальными по отношению к подпрограмме.

Переменные, описанные в подпрограмме, являются локальными для подпрограммы и в программе использоваться не могут. Если, в свою очередь, уже внутри подпрограммы описаны вложенные подпрограммы, то там разграничение аналогично.



КАК ОСУЩЕСТВЛЯЕТСЯ ПЕРЕДАЧА ДАННЫХ В ПОДПРОГРАММЫ?

ПЕРЕДАЧА ПАРАМЕТРОВ ПО ЗНАЧЕНИЮ И ПО ССЫЛКЕ

Для передачи данных в процедуры и функции в Turbo Pascal предусмотрено два механизма: передача параметров по значению и передача параметров по ссылке. В первом случае при вызове процедуры ей передаются определенные значения, которые присваиваются соответствующим переменным. В качестве передаваемых значений могут быть константы, значения каких-либо переменных и т.п. В последнем случае никакие изменения в процедуре над переданным значением не отражаются над исходной переменной, значение которой передавалось.

Помимо передачи по значению, в Turbo Pascal предусмотрена передача параметров по ссылке, когда подпрограмме передается не значение какой-либо переменной, а сама переменная (то есть соответствующая ей область памяти). При этом область памяти передаваемой переменной передается переменной в подпрограмме. Все изменения над переменной подпрограммы будут отражаться над исходной переменной. При описании подпрограмм перед переменными, значения которым должны передаваться по ссылке, ставится слово **Var** (один раз перед началом описания таких переменных). Перед переменными, значения которым передаются по значению, слово **Var** не ставится.

ГЛАВА 8.

РАБОТА С ФАЙЛАМИ



8.1. ЧТО ТАКОЕ ФАЙЛ. ИМЯ ФАЙЛА

Вся информация и данные, которые хранятся в компьютере, представлены в нем в виде файлов. Текстовые документы, программы, цифровые фотографии, базы данных и т.д. – все это хранится в виде файлов. В общем случае файл – это набор данных, который имеет свое имя.

Для упорядочивания и хранения файлов используются папки, или как их еще называют – каталоги. При этом внутри одних каталогов (папок) можно размещать другие каталоги (папки) и так далее.

Каталог (папка), находящийся внутри другого каталога (папки), является по отношению к нему вложенным. Очень удобно располагать файлы, относящиеся к определенным программам, темам и т.д., в отдельных каталогах (папках). Таким образом решается проблема хаоса, который бы был на компьютере, если бы все файлы на нем были в «одной большой куче».

Имя каждого файла имеет следующий формат:

<Имя файла>.<Расширение>

Расширение состоит из трех букв и призвано характеризовать, что это за файл: исполняемая программа, текстовый документ, цифровая фотография и т.п.

Некоторые наиболее распространенные расширения файлов приведены в табл. 8.1.

Таблица 8.1. *Некоторые наиболее распространенные расширения файлов*

Расширение	Описание
pas	Расширение файлов с исходными текстами программ на Паскале
txt, doc	Расширения текстовых документов
jpg, gif, png, tiff	Расширения фотографических изображений, хранящихся на компьютере
htm, html	Расширения интернет-страниц
Hlp	Расширение файлов справки
exe, com	Расширения выполняемых программ
rar, zip	Расширения архивных файлов
avi, mpg, mpeg, wmv	Расширения видеофайлов
Xls	Расширение электронных таблиц Excel
dll, lib, obj	Расширения файлов-библиотек

Помимо собственного имени файла принято различать еще полное имя файла, которое включает в себя полное имя логического диска, на котором расположен файла, а также последовательность каталогов (папок), начиная от самого первого, находящегося на логическом диске и не принадлежащем какому-либо другому каталогу, и до того каталога, который содержит файл. Каталоги в последовательности разделяются символами “обратный слэш”. Данная последовательность называется **путь к файлу**. Общий вид полного имени файла таков:

```
<Имя логического диска>:\<Каталог 1>\... \<Каталог n>\  
<Имя файла>.<Расширение>
```

Файлы аналогичны массивам, только информация хранится не в оперативной памяти, а на постоянном носителе. Однако имеется

существенное отличие файлов от массивов, связанное с природой этих структур данных. Описание массива задается в программе, причем, обычно, уже на этапе написания программы. Соответственно программе заранее известны типы элементов массива и их количество.

С файлами дело обстоит совершенно по-другому — для программы, файл, расположенный на каком-либо носителе, является всего лишь набором байтов, а его логическая структура определяется подпрограммой чтения файла. В качестве иллюстрации такого подхода можно привести диалог выбора формата файла в программе MS Word, когда она не может самостоятельно его определить. После выбора пользователем формата, информация будет прочитана из файла методом, соответствующим выбранному формату. Соответственно один и тот же файл можно прочитать по-разному.

Информация, находящаяся в файле может быть использована тогда, когда программа получает к нему доступ, соответственно, на этапе разработки программы размер файла не известен, вследствие чего для файлов, в отличие от массивов, реализован не индексный метод доступа (когда к конкретному элементу массива можно непосредственно обратиться по его индексу, не перебирая остальные), а последовательный, то есть, получить доступ к какому-либо элементу, можно только считав все предыдущие.

В общем виде, файл можно представить последовательностью элементов, имеющей начало и конец. Для определения элемента, который может быть прочитан в данный момент, или в который может быть записана информация, существует, так называемый, указатель текущего элемента

8.2. КЛАССИФИКАЦИЯ ФАЙЛОВ В TURBO PASCAL

Файлы разделяются на группы в соответствии с типом элементов, которые они содержат. Классификация файлов необходима для правильного выбора средств работы с ними.

По логической структуре файлы делятся на **текстовые** и **двоичные**. Текстовыми файлами называются файлы, информация в которых представлена в виде, понятном пользователю. Запись информации в текстовые файлы аналогична выводу ее на экран — вне зависимости от типов данных выводимых единиц информации, она записывается в файл в текстовом виде. Вывод информации в двоичный файл происходит в формате, принятом для хранения информации в оперативной памяти.

Двоичные файлы подразделяются на **нетипизированные** и **типизированные**. Нетипизированные файлы представляются в виде совокупности байтов, то есть каждый элемент нетипизированного файла имеет тип `Byte`. Типизированные файлы являются



Рис. 8.1. Классификация файлов

более сложными структурами, хотя на самом деле также являются совокупностями байтов, — элементы типизированных файлов могут иметь сложные типы данных, в том числе они могут являться записями.

8.3. РАБОТА С ФАЙЛАМИ В TURBO PASCAL

ОБЩАЯ МЕТОДИКА РАБОТЫ С ФАЙЛАМИ

Независимо от того, с каким именно файлом строится работа из создаваемой программы, порядок действий над ним таков:

1. В разделе описания производится описание файловой переменной, через которую программа будет обращаться с файлом
2. В разделе логики программы производится сопоставление файловой переменной файлу на диске
3. Файл открывается для чтения или для записи
4. Производится чтение или запись данных в файл
5. Заккрытие файла

ОПИСАНИЕ ФАЙЛОВОЙ ПЕРЕМЕННОЙ

Описание файловой переменной зависит от типа используемого файла. Для текстовых файлов применяются переменные типа `Text`:

```
Var
```

```
    <Имя переменной>: Text;
```

Для двоичных нетипизированных файлов применяются файловые переменные типа `File`:

```
Var
```

```
    <Имя переменной>: File;
```

Для двоичных типизированных файлов применяются переменные типа `File` с указанием типа элементов в файле после ключевого слова `Of`:

Var

<Имя переменной>: File Of <Тип элементов>;

СОПОСТАВЛЕНИЕ ФАЙЛОВОЙ ПЕРЕМЕННОЙ ФАЙЛУ НА ДИСКЕ

Сопоставление файловых переменных файлам производится независимо от типа файла. При этом для сопоставления описанной ранее файловой переменной какому-либо файлу на диске в Turbo Pascal используется процедура `Assign` (от англ. **Assign** — назначить), имеющая следующий вид:

`Assign(<Файловая переменная>, <Имя файла>);`

В качестве параметра `Файловая переменная` может использоваться переменная или константа типа `String`, содержащая полное или относительное имя файла на диске.

ОТКРЫТИЕ ФАЙЛА ДЛЯ ЧТЕНИЯ ИЛИ ДЛЯ ЗАПИСИ

Каждый файл, которому сопоставлена файловая переменная в любой момент времени выполнения программы может находиться в трех состояниях:

- **Файл закрыт:** с ним нельзя выполнять никаких действий. Данный режим устанавливается для файла при сопоставлении ему файловой переменной;
- **Файл открыт для чтения:** для файла допустима операция получения элемента, на который указывает указатель текущего элемента;
- **Файл открыт для записи:** для файла допустима операция добавления или изменения текущего элемента, на который указывает указатель текущего элемента;

Открытие файла для чтения производится процедурой `Reset` (от англ. **Reset** — сбросить) с одним параметром — файловой переменной, определяющей, какой именно файл следует открыть для чтения:

`Reset(<Файловая переменная>);`

Открытие файла для записи производится процедурой `Rewrite` (от англ. **Rewrite** — перезаписать) с одним параметром — файловой переменной, определяющей, какой именно файл следует открыть для записи:

`Rewrite(<Файловая переменная>);`

Процедуры `Reset` и `Rewrite` требуют некоторого пояснения. Если файл открывается программой для чтения, то он должен существовать на диске, то есть файловая переменная должна содержать имя реально существующего файла. Если данное условие не выполняется, то при попытке открытия несуществующего файла будет выдана ошибка `Error 2: File not found` — Ошибка 2: Файл не найден.

Процедура `Rewrite` не требует наличия файла, которому сопоставлена файловая переменная: если файл с таким именем не существует, то он будет создан. Если файловая переменная ссылается на файл с недопустимым именем, например, содержащим пробелы, то при попытке открытия его для записи будет выдана та же самая ошибка, что и при открытии несуществующего файла для чтения. Если файловая переменная ссылается на существующий файл, то **все его содержимое будет удалено** при открытии для записи.

ЧТЕНИЕ ДАННЫХ ИЗ ФАЙЛА И ЗАПИСЬ ДАННЫХ В ФАЙЛ.

ТЕКСТОВЫЕ ФАЙЛЫ

В этом разделе будут рассмотрены чтение данных из текстового файла и запись файлов в текстовый файл. Операции чтения и записи по отношению к другим типам файлов будут рассмотрены в отдельных параграфах данной главы. Там же будет дано более подробное описание самих типов файлов.

Работа с текстовыми файлами аналогична работе с экраном в текстовом режиме. Чтение и запись в них производится операторами `Read`, `ReadLn`, `Write`, `WriteLn` с указанием файловой переменной, сопоставленной файлам, в которые необходимо про-

известии вывод или ввод информации, в качестве первого параметра:

```
Read(<Файловая переменная>, <Приемник информ.1>, ...,
    <Приемник информации n>);
ReadLn(<Файловая переменная>, <Приемник информ.1>, ...,
    <Приемник информации n>);
Write(<Файловая переменная>, <Выводимая информация>,
    ..., <Выводимая информация>);
WriteLn(<Файловая переменная>, <Выводимая информация>,
    ..., <Выводимая информация>);
```

Приведем пример создания текстового файла, содержащего строку "Мой текстовый файл". Файл будет иметь имя `text1.txt` и находиться в текущем каталоге (вместе с выполняемой программой).

```
Program WriteTextFile;
Var
    f: Text;                                {описание файловой переменной с
                                            именем t для работы с текстовыми
                                            файлами}

Begin
    Assign(f, 'text1.txt');                 {сопоставление файловой
                                            переменной f файлу с именем
                                            text1.txt, находящемуся в текущем
                                            каталоге}

    Rewrite(f);                             {открытие файла, которому
                                            сопоставлена переменная
                                            f (text1.txt) для записи}

    Write(f, 'Мой текстовый файл');         {вывод строки в файл, которому
                                            сопоставлена файловая
                                            переменная f (text1.txt)}

    Close(f);                               {закрытие файла, которому
                                            сопоставлена переменная f (text1.
                                            txt)}

End.
```

Убедиться, что файл создан, можно с помощью любого текстового редактора, открыв в нем данный файл.

Приведем пример чтения информации из файла, созданного предыдущим примером. Считав строку из файла, выведем ее на экран:

```
Program ReadTextFile;  
Uses  
  CRT;  
Var  
  t: Text;           {описание файловой переменной с именем t  
                     для работы с текстовыми файлами}  
  S: String;         {описание строковой переменной, в которую  
                     будет производиться чтение из файла}  
Begin  
  ClrScr;  
  Assign(t, 'text1.txt'); {сопоставление файловой переменной t файлу  
                           с именем text1.txt, находящемуся в текущем  
                           каталоге}  
  Reset(t);           {открытие файла, которому сопоставлена  
                       переменная t (text1.txt) для чтения}  
  ReadLn(t, S);        {чтение строки из файла в переменную S}  
  WriteLn(S);          {вывод прочитанной строки на экран}  
  Close(t);            {заккрытие файла, которому сопоставлена  
                       переменная t (text1.txt)}  
  ReadKey;  
End.
```

ЗАКРЫТИЕ ФАЙЛА

Для подтверждения записанной информации в файл и для сокращения ресурсов, занятых системой для поддержания файлов в открытом состоянии, после окончания работы с файлом необходимо его закрыть — перевести в неактивное состояние, когда файл недоступен для чтения и записи. Независимо от типа файла его закрытие файла производится процедурой `Close` (от англ. `Close` — закрывать):

`Close (<Файловая переменная>);`

При открытии файла для записи, информация, содержащаяся в нем, удаляется, затем, программа заново формирует файл, и, после выполнения процедуры `Close`, файл закрывается и вся ин-

формация, записанная в него, подтверждается (остается на диске). Если программа заканчивается, а файл, в который производился вывод, не закрыт, то его длина остается нулевой, то есть информация реально не записывается в файл.

При открытии файла для чтения не может возникнуть такого рода ошибки, однако существует другая проблема. Дело в том, что количество файлов, которое одновременно может быть открыто всеми программами одновременно, ограничено. Таким образом, если, например, какая-либо процедура открывает файл, используя для этого локальную переменную, и не закрывает его после чтения информации, то такой файл считается открытым, и при некотором количестве вызовов программа аварийно завершается с ошибкой `Error 4: Too many open files` — Ошибка 4: Слишком много открытых файлов.

Таким образом, необходимо внимательно следить за состоянием файлов, и, если какой-либо из них более не понадобится для работы, его следует закрыть.

8.4. ОСОБЕННОСТИ РАБОТЫ С ФАЙЛАМИ СПЕЦИАЛЬНОГО ВИДА

Двоичные типизированные файлы

Двоичные файлы содержат информацию, аналогичную по строению информации в оперативной памяти. Чтение и запись двоичных типизированных файлов осуществляется, соответственно, процедурами `Read` и `Write` (формы `ReadLn` и `WriteLn` не используются, так как файлы являются двоичными, следовательно, не содержат в себе каких-либо строк). Тип данных, записываемых в файл, должен строго соответствовать типу, заявленному в описании файловой переменной.

```
Read(<Файловая переменная>, <Приемник информации 1>,  
    ..., <Приемник информации n>);  
Write(<Файловая переменная>, <Источник информации 1>,  
    ..., <Источник информации n>);
```


В отличие от работы с экраном и текстовыми файлами, в качестве параметров **Источник информации** должны использоваться переменные. Использование констант недопустимо.

ДВОИЧНЫЕ НЕТИПИЗИРОВАННЫЕ ФАЙЛЫ

Методы работы с двоичными нетипизированными файлами существенно отличаются от методов работы с текстовыми и двоичными типизированными файлами, являются более сложными, однако и предоставляют более развитые возможности. Рассмотрим особенности использования файлов данного типа.

1. Описание файловой переменной и сопоставление ее файлу уже рассмотрено нами.
2. Открытие файла для чтения или записи содержит дополнительный параметр, определяющий размер единицы информации (в байтах), используемой для обмена данными программы и операционной системы:

```
Reset(<файловая переменная>, <Размер единицы информации>);  
Rewrite(<файловая переменная>, <Размер единицы информации>);
```

3. Чтение информации из файла и запись информации в файл осуществляется, соответственно, процедурами **BlockRead** (от англ. **Read Block** — прочитать блок информации) и **BlockWrite** (от англ. **Write Block** — записать блок информации), имеющими следующие параметры:

```
BlockRead(<файловая переменная>, <Приемник информации>,  
          <Количество единиц информации>);  
BlockWrite(<файловая переменная>, <Источник информации>,  
           <Количество единиц информации>);
```

В качестве параметров **Приемник информации** и **Источник информации** могут использоваться переменные любого типа, в том числе и области данных, на которые ссылаются нетипизированные указатели, что позволяет сохранять в файлах спрайты. Параметр **Количество единиц информации** указывает размер читаемых

и записываемых данных в единицах информации, размер которых указан при открытии файла процедурой `Reset` или `Rewrite`.

Рекомендуется при открытии файлов указывать в качестве `Размера единицы информации` число 1, тогда при вызове процедур `BlockRead` и `BlockWrite` параметр `Количество единиц информации` будет равен размеру переменной, заданной в качестве параметра `Приемник информации` для процедуры `BlockRead` или `Источник информации` для процедуры `BlockWrite`. Размер переменной можно определить с помощью функции `SizeOf` (от англ. **Size Of...** — размер чего-либо), возвращающей размер в байтах переменной, заданной в качестве параметра:

`SizeOf (<Переменная>)`

4. закрытие файла производится обычным образом — вызовом процедуры `Close`.

8.5. ВСПОМОГАТЕЛЬНЫЕ ОПЕРАЦИИ НАД ФАЙЛАМИ

ОПРЕДЕЛЕНИЕ РАЗМЕРА ФАЙЛА

Как правило программа не располагает данными о количестве информации, размещенной в файле, с которым она работает. Поэтому, когда производится считывание данных из файла, должна производиться постоянная проверка «а не закончился ли файл?». Для этого в Turbo Pascal предусмотрена функция `EOF` (от англ. **End Of File** — конец файла), возвращающая истинное значение, если достигнут конец файла, заданный в качестве параметра, или ложное, если конец файла не достигнут:

`EOF (<Файловая переменная>)`

Очень удобно реализовать считывание содержимого файла можно реализовать с помощью функции `EOF` условного оператора цикла `While`. При этом в теле цикла заикливается фрагмент

кода, читающий очередную строку (число) из файла, а в качестве условия цикла используется NOT Eof (<файловая переменная>). Таким образом, цикл автоматически закончится при достижении конца файла.

```

Program ReadTemps;
Uses
  CRT;
Var
  T, Sum: Real;
  TNumber: Integer;
  f: Text;
Begin
  ClrScr;
  Assign(f, 'temps.txt');           {сопоставление файловой
                                     {переменной файлу temps.txt}

  Reset(f);                         {открытие файла для чтения}

  Sum := 0;                          {сумма прочитанных чисел}

  TNumber := 0;                     {счетчик прочитанных чисел}

  While not EOF(f) do               {условный оператор цикла,
                                     {условие продолжения — "не
                                     {достигнут конец файла"}}

    begin
      ReadLn(f, T);                 {чтение из файла одного числа в
                                     {переменную T}

      Sum := Sum + T;               {увеличение суммы}

      TNumber := TNumber + 1;       {увеличение счетчика}

    end;

  Close(f);                         {заккрытие файла}

  WriteLn('Средняя температура: ', Sum / TNumber :0:3);

  ReadKey;

End.

```

Для двоичных файлов предусмотрен еще один способ определения длины файла, часто более удобный в использовании, чем функция EOF. Процедура FileSize (от англ. **Size Of File** —

размер файла) возвращает размер файла, заданный ей в качестве параметра:

FileSize (<Файловая переменная>)

Для двоичных типизированных файлов данная функция возвращает количество элементов в файле, а для двоичных нетипизированных — количество единиц информации, используемых для обмена программы с операционной системой, размер которых задается при открытии файла для чтения или записи с помощью, соответственно, процедур `Reset` и `Rewrite`.

В момент использования функции `FileSize`, файл, соответствующий параметру `Файловая переменная` должен быть открыт.

```
Program ReadTemps;
Uses
  CRT;
Var
  T, Sum: Real;
  i: Integer;
  f: File Of Real;
Begin
  ClrScr;
  Assign(f, 'temps.txt');
  Reset(f);
  Sum := 0;
  For i:=1 To FileSize(f) Do
    begin
      Read(f, T);
      Sum := Sum + T;
    end;
```

{сопоставление файловой
переменной файлу temps.txt}

{открытие файла для чтения}

{сумма прочитанных из файла
чисел}

{безусловный цикл по
количеству элементов в файле}

{чтение из файла одного числа в
переменную T}

{увеличение суммы прочитанных
чисел}


```
WriteLn('Средняя температура: ', Sum / FileSize(f) :0:3);  
Close(f);                               {закрытие файла}  
ReadKey;  
End.
```

ИЗМЕНЕНИЕ ТЕКУЩЕГО ЭЛЕМЕНТА

Для двоичных файлов предусмотрена операция изменения текущего элемента. Таким образом, можно открыть файл для чтения или записи, в результате чего текущим элементом становится первый, а, затем, не считывая элементы переместить указатель текущего элемента в любое место.

Seek (<Файловая переменная>, <Номер нового текущего элемента>);

Процедура **Seek** (от англ. **Seek** — искать) перемещает указатель текущего элемента файла, заданного параметром Файловая переменная, в позицию заданную параметром Номер текущего элемента. Нумерация элементов файла ведется от нуля, то есть первый элемент имеет номер 0.

Приведем пример программы, считывающей первый и последний элементы файла без считывания элементов, расположенных между ними:

```
Program RandomRead;  
Uses  
  CRT;  
Var  
  T, Sum: Real;  
  f: File Of Real;  
Begin  
  ClrScr;  
  Assign(f, 'temps.bin');  
  Reset(f);                               {открытие файла для чтения}  
  Seek(f, 0);                             {позиционирование указателя  
                                           текущего элемента на первый  
                                           элемент}  
  
  Read(f, T);                             {чтение текущего элемента  
                                           (первого)}
```



```
WriteLn('Первый элемент: ', T:0:3);
Seek(f, FileSize(f) - 1);           {позиционирование указателя
                                     текущего элемента на
                                     последний элемент}

Read(f, T);                         {чтение текущего элемента}

WriteLn('Последний элемент: ',
T:0:3);
Close(f);                           {закрытие файла}
ReadKey;
End.
```

8.6. ДОБАВЛЕНИЕ ИНФОРМАЦИИ В ФАЙЛЫ

ДОБАВЛЕНИЕ ИНФОРМАЦИИ В ТЕКСТОВЫЕ ФАЙЛЫ

Для текстовых файлов предусмотрена возможность добавления информации в их конец. Для этого файл следует открывать не процедурой `Rewrite`, а процедурой `Append` (от англ. **Append** — добавить):

Append(<Файловая переменная>);

После вызова данной процедуры файл открывается для записи, но информация в нем не стирается, как в случае использования процедуры `Rewrite`. Указатель текущего элемента перемещается в конец файла, соответственно, выводимая информация дописывается к открытому файлу.

В качестве примера, допишем к файлу, созданному программой `WriteTextFile`, еще одну строку: "Состоит из двух строк":

```
Program WriteTextFile2;
Var
  t: Text;                       {описание файловой переменной}
Begin
  Assign(t, 'text1.txt');        {сопоставление файловой
                                  переменной t файлу с именем
                                  text1.txt, находящемуся в
                                  текущем каталоге}
```



```

Append(t);           {открытие файла для добавления
                      информации}

WriteLn(t);
Write(t, 'Состоит из двух строк'); {вывод строки в конец файла,
                                   которому сопоставлена
                                   файловая переменная t (text1.
                                   txt)}

Close(t);           {закрытие файла}
End.

```

В результате выполнения такой программы, файл `text1.txt` будет содержать две строки: "Мой первый текстовый файл" и "Состоит из двух строк".

ДОБАВЛЕНИЕ ИНФОРМАЦИИ В ДВОИЧНЫЕ ФАЙЛЫ

Двоичные файлы поддерживают одновременное чтение и запись элементов, причем, если производится запись элементов за пределы файла, то перед записываемым элементом автоматически добавляются соответствующее количество нулевых элементов.

Напишем программу, записывающую информацию в десятый элемент двоичного типизированного файла с элементами типа `Integer`:

```

Program RandomWrite;
Uses
  CRT;
Var
  T: Real;
  i: Integer;
  F: File Of Real;
Begin
  Assign(F, 'rand.bin'); {сопоставление файловой переменной файлу
                          rand.bin, файл считается текстовым}

  Rewrite(F);           {открытие файла temps.bin для записи}

  Seek(f, 9);           {перемещение указателя текущего элемента на
                          десятый}

  T := 20.5;

```



```
Write(F, T);           {запись значения в десятый элемент файла,
                        {предыдущие девять заполняются нулями}

Close(F);               {закрытие файла}

ReadKey;

End.
```

Добавление элементов в двоичные файлы происходит аналогичным образом — открытием файла, изменением текущего элемента, и записью в него информации. Для того, чтобы не удалялась предыдущая информация, файл необходимо открывать процедурой `Reset`:

```
Program AddBin;
Uses
  CRT;
Var
  T: Real;
  i: Integer;
  F: File Of Real;
Begin
Assign(F, 'rand.bin'); {сопоставление файловой переменной файлу
                        {rand.bin, файл считается текстовым}

Reset(F);              {открытие файла temps.bin для записи}

Seek(f, 10);           {перемещение указателя текущего элемента
                        {на одиннадцатый}

T := 20.5;

Write(F, T);           {запись значения в одиннадцатый элемент
                        {файла, остальные остаются без изменения}

Close(F);              {закрытие файла}

ReadKey;

End.
```

8.7 ОПЕРАЦИИ С ФАЙЛАМИ И КАТАЛОГАМИ

Для эффективного управления расположением файлов в Turbo Pascal предусмотрено множество полезных процедур:

- `Erase(<Файловая переменная>)` — удаляет файл, соответствующий указанной файловой переменной. При этом данный файл не должен быть открыт для чтения или

записи. после удаления файл не следует закрывать, так как он не был открыт.

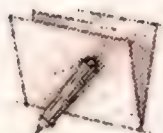
- Переименование и перенос файлов на диске производится процедурой **Rename** (от англ. **Rename** — переименовать), имеющей следующий вид **Rename (<Файловая переменная>, <Новое имя файла>)** — с помощью этой процедуры можно переименовать или перенести файл в другое место на диске. В качестве параметра Новое имя файла используется строковая переменная или константа, содержащая новое имя файла, при этом возможна смена каталога, в котором находился файл, в результате чего и осуществляется не просто переименование файла, а его перенос. Изменять положение файла таким образом, чтобы он оказался на другом логическом диске не допускается.
- **MkDir (<Имя каталога>)** — процедура создания каталога. Параметр Имя каталога является строковой переменной или константой, содержащей название каталога по правилам, аналогичным именам файлов. Если задается полное название каталога, то он появляется на диске, если его еще не было. Возможно задание относительного названия каталога. При попытке создания уже существующего каталога, информация в нем не стирается и выдается ошибка времени выполнения программы **Error 5: File access denied** — Ошибка 5: Запрещен доступ к файлу.
- **Rmdir (<Имя каталога>)** — удаляет каталог. При этом удаление возможно только для каталогов, Удаление возможно только для каталогов, которые не содержат файлов или подкаталогов.

ШПАРГАЛКА. КОНТРОЛЬНЫЕ ВОПРОСЫ И ОТВЕТЫ ПО ТЕМЕ



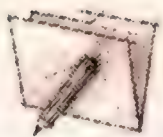
Что такое файл?

Вся информация и данные, которые хранятся в компьютере, представлены в нем в виде файлов. Текстовые документы, программы, цифровые фотографии, базы данных и т.д. – все это хранится в виде файлов. Строго говоря, файл – это поименованная область на диске, имеющая свое имя.



На какие типы подразделяются файлы с точки зрения Turbo Pascal?

В Turbo Pascal принято разделять файлы по следующим типам: текстовые, двоичные типизированные и двоичные нетипизированные. Тип файла определяет методику и операторы работы с ним.



Какова стандартная методика работы с файлами в Turbo Pascal и какие операторы для этого предусмотрены?

Прежде чем работать с файлом, в программе должна быть описана переменная файлового типа и поставлена в соответствие с файлом. Для последнего в Turbo Pascal предусмотрен оператор **Assign**. Все действия с файлом осуществляются через файловую переменную.

Работа с файлами заключается либо в чтении информации из них, либо в записи информации в них. В соответствии с этим файл должен быть открыт либо для чтения – процедура **reset**, либо для записи – процедура **rewrite**. Чтение и запись информации производится стандартными процедурами **write**, **writeln**, **read**, **readln**. Для двоичных типизированных файлов – процедурами **BlockWrite** и **BlockRead**.

По завершении работы с файлом (либо для перехода между режимами чтения-записи) он должен быть закрыт с помощью процедуры **Close**.

UNIVERSITY OF MICHIGAN
LIBRARY

1964

THE UNIVERSITY OF MICHIGAN
LIBRARY

THE UNIVERSITY OF MICHIGAN
LIBRARY

THE UNIVERSITY OF MICHIGAN
LIBRARY

THE UNIVERSITY OF MICHIGAN
LIBRARY

THE UNIVERSITY OF MICHIGAN
LIBRARY

THE UNIVERSITY OF MICHIGAN
LIBRARY

ГЛАВА 9.

МОДУЛИ



9.1. ЧТО ТАКОЕ МОДУЛЬ (БИБЛИОТЕКА) ПОДПРОГРАММ

Для более удобного использования накопленного объема подпрограмм возможно объединение их в **библиотеки (модули)** — отдельные файлы с промежуточным программным кодом (объектные файлы), подключаемые к программе в случае необходимости в разделе подключения модулей `Uses`. Обычно, процедуры и функции, включаемые в библиотеки, обладают общей функциональной направленностью, то есть близки друг другу по выполняемым задачам.

В качестве примера можно привести стандартный модуль `CRT`, в который включены процедуры и функции, осуществляющие какие-либо действия с клавиатурой и экраном в текстовом режиме. Еще один пример модуля — стандартный модуль `Graph`, содержащий процедуры и функции работы с экраном в графическом режиме.

Исходный текст модуля располагается в отдельном файле на диске, и содержит три основных секции:

- Интерфейсная секция, в которой располагаются заголовки процедур и функций, составляющих модуль.
- Описательная секция, в которой располагаются описания процедур и функций, заголовки которых указаны в интерфейсной секции, а также другие процедуры и

функции, используемые подпрограммами данного модуля, но не доступные программам, подключающим модуль.

- Секция инициализации (не обязательна), содержащая команды, которые необходимо выполнить при подключении модуля к программе. Таким образом, перед выполнением какой-либо программы, к которой подключаются внешние модули, сначала выполняются инициализационные части модулей (в порядке подключения).

9.2. СТРУКТУРА МОДУЛЯ В TURBO PASCAL 7.0

Модуль имеет следующую структуру:

Unit <Название>;	Заголовок модуля
Interface	Указание на начало интерфейсной секции
Uses	Раздел подключения модулей
Const	Раздел описания констант
Type	Раздел описания типов данных
Var	Раздел описания переменных
Procedure	Раздел описания заголовков процедур и функций
Function	
Implementation	Указание на начало описательной секции
Uses	Раздел подключения модулей
Label	Раздел описания меток безусловного перехода
Const	Раздел описания констант
Type	Раздел описания типов данных
Var	Раздел описания переменных

Procedure	Раздел описания процедур и функций
.....	
Function	
.....	
begin	Начало раздела инициализации (не обязательно)
.....	Раздел описания логики инициализации
end;	Окончание раздела инициализации

Заголовок модуля, в отличие от заголовка программы должен указываться обязательно, так как название модуля используется программой, которая его подключает для получения доступа к описанным в нем процедурам и функциям. Имя файла с исходным текстом модуля должно быть таким же, как название модуля, поэтому название модуля не должно превышать восьми символов — системного ограничения MS-DOS на длину имен файлов.

ВМЕСТО ЗАКЛЮЧЕНИЯ. ГРАФИКА И ГРАФИЧЕСКИЙ РЕЖИМ В TURBO PASCAL

Для работы с графикой в Turbo Pascal предусмотрен специальный графический режим. В графическом режиме экран имеет размеры 640x480: по горизонтали от 0 до 639, начиная с левого верхнего угла, по вертикали – от 0 до 479, начиная с левого верхнего угла. Процедуры работы с графикой определены в модуле **Graph**, который необходимо будет подключить.

Для перехода в графический режим используется процедура **InitGraph**, а для выхода из графического режима – процедура **CloseGraph**. В качестве параметров процедуры **InitGraph** указываются параметры экрана и графического драйвера. Чтобы не вдаваться в детали этого процесса, сразу приведем, как это должно выглядеть:

```
Program GraphMode;
Uses
  Graph;
Var
  GraphDriver, GraphMode: Integer;
Begin
  GraphDriver := Detect;
  GraphMode := 0;
  InitGraph(GraphDriver, GraphMode, '');

  {.....операторы.....}
  {.....операторы.....}

  CloseGraph
End.
```

Используемый режим никак не влияет на методику использования условных операторов, циклов и т.п. Различия наблюдаются только лишь в выводе информации на экран, а также в наличии большого количества дополнительных процедур рисования, предусмотренных для графического режима и отсутствующих в режиме текстовом.

Описание процедур графического режима приведено в приложении. В качестве примера рассмотрим две программы. Первая программа рисует квадрат:

```
Program Kvadrat;  
Uses  
  Graph;  
Var  
  GraphDriver, GraphMode: Integer;  
Begin  
  GraphDriver := Detect;  
  GraphMode := 0;  
  InitGraph(GraphDriver, GraphMode, '');  
  
  line(100, 100, 200, 100); line(200, 100, 200, 200);  
  line(100, 200, 100, 100); line(200, 200, 100, 200);  
  
  CloseGraph;  
End.
```

Вторая программа выводит сообщение в произвольном месте экрана и заставляет его переливаться разными цветами:

```
Program TextGraph;  
Uses  
  Graph;  
Var  
  GraphDriver, GraphMode: Integer;  
  i: Integer;  
Begin  
  GraphDriver := Detect;  
  GraphMode := 0;  
  InitGraph(GraphDriver, GraphMode, '');  
  
  for i:=1 to 15 do  
    begin  
      setcolor(i);  
      outtextxy (200, 200, 'Текст в графическом режиме')  
      delay (10000); {задержка}  
    end;  
  CloseGraph;  
End.
```


ПРИЛОЖЕНИЯ



ПРИЛОЖЕНИЕ 1. ПОДПРОГРАММЫ МОДУЛЯ CRT

Модуль CRT содержит подпрограммы вывода информации на экран в текстовом режиме и установки различных видов текстовых режимов, а также поддерживает ввод информации с клавиатуры.

ПРОЦЕДУРА AssignCRT

Синтаксис: Procedure AssignCrt (Var f: Text)

Описание: позволяет связать файловую переменную типа Text с экраном в текстовом режиме. Процедура аналогична процедуре Assign, за исключением того, что имя файла, с которым связывается файловая переменная, не указывается. Вся информация, выводимая в файл, отображается на экране, а информация, читаемая из файла, на самом деле должна быть введена с клавиатуры. Перед использованием процедур Read, ReadLn, Write и WriteLn необходимо открыть файл для чтения или записи процедурами Reset и ReWrite.

В каком режиме используется: в текстовом.

ПРОЦЕДУРА ClrEol

Синтаксис: Procedure ClrEol

Описание: стирает все символы с экрана из текущего положения каретки до конца строки. Положение каретки не изменяется. При стирании символов используются текущие настройки цвета символов и фона, установленные с помощью процедур TextColor и TextBackground. При определении конца строки учитывается положение активного окна вывода, установленное процедурой Window.

В каком режиме используется: в текстовом.

ПРОЦЕДУРА CLRSCR

Синтаксис: Procedure ClrScr

Описание: стирает все символы с экрана с учетом положения активного окна вывода, установленного процедурой Window. Каретка перемещается в левый верхний угол активного окна, при стирании символов используются текущие настройки цвета символов и фона, установленные с помощью процедур TextColor и TextBackground.

В каком режиме используется: в текстовом.

ПРОЦЕДУРА DELAY

Синтаксис: Procedure Delay(Ms: Word)

Описание: осуществляет приостановку программы на время, заданное в качестве параметра. Время задается в миллисекундах.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА DELLINE

Синтаксис: Procedure DelLine

Описание: стирает все символы строки экрана, в которой находится каретка. Положение каретки не изменяется, а все строки, расположенные ниже стираемой строки, смещаются на одну строку вверх. Освободившаяся нижняя строка заполняется пробелами. При заполнении нижней строки используются текущие настройки цвета символов и фона, установленные с помощью процедур TextColor и TextBackground.

В каком режиме используется: в текстовом.

ПРОЦЕДУРА GotoXY

Синтаксис: Procedure (X, Y: Integer)

Описание: устанавливает каретку в положение, заданное параметрами. При изменении положения каретки учитывается положение активного окна вывода, установленное процедурой Window.

В каком режиме используется: в текстовом.

ПРОЦЕДУРА HighVideo

Синтаксис: Procedure HighVideo

Описание: изменяет поведение процедуры TextColor, заставляя ее использовать вместо цветов с номерами от 0 до 7 цвета с номерами, соответственно от 8 до 15 (более яркие).

В каком режиме используется: в текстовом.

ПРОЦЕДУРА DelLine

Синтаксис: Procedure DelLine

Описание: вставляет пустую строку на экране в положение каретки. Все строки, расположенные ниже стираемой строки, смещаются на одну строку вниз. Новая строка заполняется пробелами. При заполнении строки используются текущие настройки цвета символов и фона, установленные с помощью процедур TextColor и TextBackground.

В каком режиме используется: в текстовом.

ФУНКЦИЯ KeyPressed

Синтаксис: Function KeyPressed: Boolean

Описание: определяет, нажата ли клавиша, значение которой еще не прочитано функцией `ReadKey`.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА `LowVideo`

Синтаксис: `Procedure LowVideo`

Описание: изменяет поведение процедуры `TextColor`, заставляя ее использовать вместо цветов с номерами от 8 до 15 цвета с номерами, соответственно от 0 до 7 (менее яркие).

В каком режиме используется: в текстовом.

ПРОЦЕДУРА `NormVideo`

Синтаксис: `Procedure NormVideo`

Описание: восстанавливает поведение процедуры `TextColor`, отменяя режим установленный процедурами `HighVideo` и `LowVideo`. После вызова процедуры будет использоваться режим, установленный до запуска программы.

В каком режиме используется: в текстовом.

ПРОЦЕДУРА `NoSound`

Синтаксис: `Procedure NoSound`

Описание: выключает звук, включенный процедурой `Sound`.

В каком режиме используется: в текстовом, графическом.

Функция `ReadKey`

Синтаксис: `Function ReadKey`

Описание: Возвращает символ, сопоставляемый клавише, нажатой на клавиатуре.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА `SOUND`

Синтаксис: `Procedure Sound(Hz: Word)`

Описание: включает звук на внутреннем динамике (PC Speaker) с частотой (в Герцах), заданной в качестве параметра. Соответствие частот нотам основной октавы приведено в табл. П1. Звук, включенный данной процедурой, самостоятельно не прекращается, его выключение производится процедурой `NoSound`. Для получения некоторой длительности звука можно использовать процедуру `Delay`.

В каком режиме используется: в текстовом, графическом.

Таблица П1. *Соответствие частот нотам*

Нота	Частота (Гц)
Си	247
До	262
Ре	294
Ми	330
Фа	350
Соль	392
Ля	440
Си	494

ПРОЦЕДУРА `TEXTBACKGROUND`

Синтаксис: `Procedure TextBackground(Color: Byte)`

Описание: устанавливает текущий цвет фона, заданный в качестве параметра.

В каком режиме используется: в текстовом.

ПРОЦЕДУРА TEXTCOLOR

Синтаксис: Procedure TextColor(Color: Byte)

Описание: устанавливает текущий цвет символов, заданный в качестве параметра. При добавлении к номеру цвета константы Blink модуля CRT (значение 128) выводимые символы будут мигать.

В каком режиме используется: в текстовом.

ПРОЦЕДУРА TEXTMODE

Синтаксис: Procedure TextMode(Mode: Integer)

Описание: устанавливает текстовый режим, отличный от стандартного. Режим задается параметром и может иметь значения, приведенные в табл. П2. Существует возможность увеличения количества строк для каждого режима, с помощью добавления константы Font8x8.

В каком режиме используется: в текстовом, графическом.

Таблица П2. Текстовые режимы

Режим (строки, символы, цветность)	Константа
40x25 черно-белый	BW40
40x25 цветной	CO40
80x25 черно-белый	BW80
80x25 цветной	CO80
80x25 черно-белый	Mono

ФУНКЦИЯ WHEREX

Синтаксис: Function WhereX

Описание: возвращает горизонтальную координату каретки.

В каком режиме используется: в текстовом.

Функция WHEREY

Синтаксис: Function WhereY

Описание: возвращает вертикальную координату каретки.

В каком режиме используется: в текстовом.

Процедура WINDOW

Синтаксис: Procedure Window(X1, Y1, X2, Y2: Byte)

Описание: устанавливает область отсечения выводимой информации. Вся информация, выводимая за пределы виртуального окна, координаты которого заданы параметрами, установленного данной процедурой игнорируется. Влияет на работу следующих подпрограмм модуля CRT: ClrEol, ClrScr, DelLine, GotoXY, InsLine, WhereX, WhereY, Read, Readln, Write, Writeln.

В каком режиме используется: в текстовом.

ПРИЛОЖЕНИЕ 2. ПОДПРОГРАММЫ МОДУЛЯ GRAPH

Модуль Graph содержит подпрограммы вывода информации на экран в графическом режиме и установки различных видов графических режимов.

Процедура ARC

Синтаксис: Procedure Arc(X, Y: Integer; StAngle, EndAngle, Radius; Word)

Описание: выводит дугу окружности.

В каком режиме используется: в графическом.

Процедура BAR

Синтаксис: Procedure Bar(x1, y1, x2, y2: Integer)

Описание: выводит закрашенный прямоугольник.

В каком режиме используется: в графическом.

ПРОЦЕДУРА BAR3D

Синтаксис: Procedure Bar3D(x1, y1, x2, y2: Integer; Depth: Word; Top: Boolean)

Описание: выводит параллелепипед.

В каком режиме используется: в графическом.

ПРОЦЕДУРА CIRCLE

Синтаксис: Procedure Circle(X, Y: Integer; Radius: Word)

Описание: выводит окружность.

В каком режиме используется: в графическом.

ПРОЦЕДУРА CLEARDEVICE

Синтаксис: Procedure ClearDevice

Описание: удаляет информацию с экрана полностью.

В каком режиме используется: в графическом.

ПРОЦЕДУРА CLEARVIEWPORT

Синтаксис: Procedure ClearViewport

Описание: удаляет информацию из области отсечения, установленной процедурой SetViewport.

В каком режиме используется: в графическом.

ПРОЦЕДУРА CLOSEGRAPH

Синтаксис: Procedure CloseGraph

Описание: закрывает графический режим, устанавливая предыдущий текстовый, который был до инициализации графического.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА DETECTGRAPH

Синтаксис: Procedure DetectGraph (Var GraphDriver, GraphMode: Integer)

Описание: определяет допустимые для текущего аппаратного обеспечения номер драйвера и номер режима, которые можно использовать в дальнейшем при вызове процедуры InitGraph.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА DRAWPOLY

Синтаксис: Procedure DrawPoly (NumPoints: Word; Var PolyPoints)

Описание: выводит многоугольник, заданный вершинами.

В каком режиме используется: в графическом.

ПРОЦЕДУРА ELLIPSE

Синтаксис: Procedure Ellipse (X, Y: Integer;
StAngle, EndAngle: Word;
XRadius, YRadius: Word)

Описание: выводит эллипс.

В каком режиме используется: в графическом.

ПРОЦЕДУРА FILL ELLIPSE

Синтаксис: Procedure FillEllipse (X, Y: Integer;
XRadius, YRadius: Word)

Описание: выводит закрашенный эллипс.

В каком режиме используется: в графическом.

ПРОЦЕДУРА FILL POLY

Синтаксис: Procedure FillPoly (NumPoints: Word;
Var PolyPoints)

Описание: выводит закрашенный многоугольник.

В каком режиме используется: в графическом.

ПРОЦЕДУРА FLOOD FILL

Синтаксис: Procedure FloodFill (X, Y: Integer;
Border: Word)

Описание: закрашивает замкнутую область экрана, начиная из заданной точки.

В каком режиме используется: в графическом.

ПРОЦЕДУРА GET ARC COORDS

Синтаксис: Procedure GetArcCoords (Var
ArcCoords: ArcCoordsType)

Описание: возвращает параметры последней выведенной дуги в переменную, заданную в качестве параметра. Параметр имеет тип ArcCoordsType, описанный как:

```
ArcCoordsType = Record
    X, Y, Xstart, Ystart, Xend, Yend: Integer;
end;
```

В каком режиме используется: в графическом.

ПРОЦЕДУРА **GETASPECTRATIO**

Синтаксис: Procedure GetAspectRatio (Var Xasp,
Yasp: Word)

Описание: возвращает два числа, по которым можно определить соотношение ширины и высоты (в точках) экрана в данном графическом режиме.

В каком режиме используется: в графическом.

ФУНКЦИЯ **GETBKCOLOR**

Синтаксис: Function GetBkColor: Word

Описание: возвращает цвет фона.

В каком режиме используется: в графическом.

ФУНКЦИЯ **GETCOLOR**

Синтаксис: Function GetColor: Word

Описание: возвращает текущий цвет выводимой информации.

В каком режиме используется: в графическом.

ФУНКЦИЯ **GETDEFAULTPALETTE**

Синтаксис: Function GetDefaultPalette (Var
Palette: PaletteType)

Описание: возвращает информацию о цветовых составляющих системной палитры, установленной при инициализации графического режима. Информация записывается в переменную, заданную параметром, имеющим тип PaletteType, описанный в модуле Graph следующим образом:

Const


```
MaxColors = 15;  
Type  
  PaletteType = Record  
    Size : Byte;  
    Colors: Array[0..MaxColors] Of Shortint;  
  end;
```

В каком режиме используется: в графическом.

ФУНКЦИЯ GETDRIVERNAME

Синтаксис: Function GetDriverName: String

Описание: возвращает название активного в данный момент графического драйвера.

В каком режиме используется: в графическом.

ПРОЦЕДУРА GETFILLPATTERN

Синтаксис: Procedure GetFillPattern (Var
FillPattern: FillPatternType)

Описание: заносит характеристики образца шаблона заливки в переменную типа FillPatternType, описанного в модуле Graph следующим образом:

```
FillPatternType = Array [1..8] of Byte;
```

В каком режиме используется: в графическом.

ПРОЦЕДУРА GETFILLSETTINGS

Синтаксис: Procedure GetFillSettings (Var
FillInfo: FillSettingsType)

Описание: заносит характеристики заливки в переменную типа FillSettingsType, описанного в модуле Graph следующим образом:

```
FillSettingsType = Record  
  Pattern: Word;
```



```
Color : Word;  
end;
```

В каком режиме используется: в графическом.

Функция GETGRAPHMODE

Синтаксис: Function GetGraphMode: Integer

Описание: возвращает номер активного графического режима.

В каком режиме используется: в графическом.

ПРОЦЕДУРА GETIMAGE

Синтаксис: Procedure GetImage(x1, y1, x2,
y2: Integer; Var BitMap)

Описание: создает спрайт в памяти.

В каком режиме используется: в графическом.

ПРОЦЕДУРА GETLINESETTINGS

Синтаксис: Procedure GetLineSettings(Var
LineInfo: LineSettingsType)

Описание: заносит характеристики обводки в переменную типа LineSettingsType, описанного в модуле Graph следующим образом:

```
LineSettingsType = Record  
    LineStyle: Word;  
    Pattern : Word;  
    Thickness: Word;  
end;
```

В каком режиме используется: в графическом.

Функция GETMAXCOLOR

Синтаксис: Function GetMaxColor: Word

Описание: возвращает максимально допустимое количество цветов для активного графического режима.

В каком режиме используется: в графическом.

Функция **GetMaxMode**

Синтаксис: Function GetMaxMode: Integer

Описание: возвращает максимально допустимый режим для активного графического драйвера.

В каком режиме используется: в графическом.

Функция **GetMaxX**

Синтаксис: Function GetMaxX: Integer

Описание: возвращает максимально допустимую горизонтальную координату для активного графического режима.

В каком режиме используется: в графическом.

Функция **GetMaxY**

Синтаксис: Function GetMaxY: Integer

Описание: возвращает максимально допустимую вертикальную координату для активного графического режима.

В каком режиме используется: в графическом.

Функция **GetModeName**

Синтаксис: Function GetModeName (ModeNumber: Integer): String

Описание: возвращает название текущего графического режима.

В каком режиме используется: в графическом.

ПРОЦЕДУРА GetModeRange

Синтаксис: Procedure GetModeRange (GraphDriver: Integer; Var LoMode, HiMode: Integer)

Описание: возвращает минимально и максимально возможные номера графических режимов для загруженного графического драйвера.

В каком режиме используется: в графическом.

ПРОЦЕДУРА GetPalette

Синтаксис: Procedure GetPalette (Var Palette: PaletteType)

Описание: возвращает информацию о цветовых составляющих текущей палитры. Информация записывается в переменную, заданную параметром, имеющим тип PaletteType.

В каком режиме используется: в графическом.

ФУНКЦИЯ GetPaletteSize

Синтаксис: Function GetPaletteSize: Integer

Описание: возвращает размер палитры для активного графического режима.

В каком режиме используется: в графическом.

ФУНКЦИЯ GetPixel

Синтаксис: Function GetPixel (X, Y: Integer): Word

Описание: возвращает цвет точки, заданной координатами.

В каком режиме используется: в графическом.

ПРОЦЕДУРА GETTEXTSETTINGS

Синтаксис: Procedure GetTextSettings (Var
TextInfo: TextSettingsType)

Описание: заносит характеристики вывода текста, установленные процедурой SetTextStyle, в переменную типа TextSettingsType, описанного в модуле Graph следующим образом:

```
TextSettingsType = Record
    Font      : Word;
    Direction: Word;
    CharSize  : Word;
    Horiz     : Word;
    Vert      : Word;
end;
```

В каком режиме используется: в графическом.

ПРОЦЕДУРА GETVIEWSETTINGS

Синтаксис: Procedure GetViewSettings (Var
ViewPort: ViewPortType)

Описание: заносит характеристики области отсечения, установленной процедурой SetViewPort, в переменную типа ViewPortType, описанного в модуле Graph следующим образом:

```
ViewPortType = Record
    x1, y1, x2, y2: Integer;
    Clip          : Boolean;
end;
```

В каком режиме используется: в графическом.

ФУНКЦИЯ GETX

Синтаксис: Function GetX: Integer

Описание: возвращает горизонтальную координату текущего указателя.

В каком режиме используется: в графическом.

Функция GetY

Синтаксис: Function GetY: Integer

Описание: возвращает вертикальную координату текущего указателя.

В каком режиме используется: в графическом.

Процедура GRAPHDEFAULTS

Синтаксис: Procedure GraphDefaults

Описание: восстанавливает все характеристики вывода информации (цвета, стили, положение текущего указателя, и так далее) в состояние, в котором они находились при инициализации графического режима.

В каком режиме используется: в графическом.

Функция GRAPHErrorMsg

Синтаксис: Function GraphErrorMsg (ErrorCode: Integer): String

Описание: возвращает описание ошибки графической системы по заданному номеру, который можно получить с помощью функции GraphResult.

В каком режиме используется: в графическом.

Функция GRAPHRESULT

Синтаксис: Function GraphResult: Integer

Описание: возвращает номер последней ошибки графической системы после выполнения следующих подпрограмм: Bar, Bar3D, ClearViewPort, CloseGraph, DetectGraph, DrawPoly, FillPoly, FloodFill, GetGraphMode, ImageSize, InitGraph, InstallUserDriver, InstallUserFont, PieSlice, RegisterBGIdriver, RegisterBGIfont, SetAllPalette, SetFillPattern, SetFillStyle, SetGraphBufSize, SetGraphMode, SetLineStyle, SetPalette, SetTextJustify, SetTextStyle. В случае отсутствия ошибки возвращается 0.

В каком режиме используется: в графическом.

Функция IMAGE_SIZE

Синтаксис: Function ImageSize(x1, y1, x2, y2: Integer): Word

Описание: возвращает объем памяти, необходимый для хранения спрайта, имеющего размеры. Заданные параметрами.

В каком режиме используется: в графическом.

ПРОЦЕДУРА INITGRAPH

Синтаксис: Procedure InitGraph(Var GraphDriver: Integer; Var GraphMode: Integer; PathToDriver: String)

Описание: устанавливает графический режим.

В каком режиме используется: в текстовом, графическом.

Функция INSTALLUSERDRIVER

Синтаксис: Function InstallUserDriver(Name: String; AutoDetectPtr: Pointer): Integer

Описание: загружает графический драйвер в систему.

В каком режиме используется: в графическом.

Функция INSTALLUSERFONT

Синтаксис: `Function InstallUserFont(FontFileName: String): Integer`

Описание: загружает шрифт в систему.

В каком режиме используется: в графическом.

ПРОЦЕДУРА LINE

Синтаксис: `Procedure Line(x1, y1, x2, y2: Integer)`

Описание: выводит линию.

В каком режиме используется: в графическом.

ПРОЦЕДУРА LINEREL

Синтаксис: `Procedure LineRel(Dx, Dy: Integer)`

Описание: выводит линию смещением текущего указателя на заданное расстояние от его текущего положения.

В каком режиме используется: в графическом.

ПРОЦЕДУРА LINETO

Синтаксис: `Procedure LineTo(X, Y: Integer)`

Описание: выводит линию смещением текущего указателя в заданную точку.

В каком режиме используется: в графическом.

ПРОЦЕДУРА MOVEREL

Синтаксис: `Procedure MoveRel(Dx, Dy: Integer)`

Описание: перемещает текущий указатель без вывода линии на заданное расстояние от его текущего положения.

В каком режиме используется: в графическом.

ПРОЦЕДУРА MoveTo

Синтаксис: Procedure MoveTo(X, Y: Integer)

Описание: перемещает текущий указатель без вывода линии в заданную точку.

В каком режиме используется: в графическом.

ПРОЦЕДУРА OutText

Синтаксис: Procedure OutText(TextString: String)

Описание: выводит текст из позиции текущего указателя.

В каком режиме используется: в графическом.

ПРОЦЕДУРА OutTextXY

Синтаксис: Procedure OutTextXY(X, Y: Integer; TextString: String)

Описание: выводит текст из заданной точки.

В каком режиме используется: в графическом.

ПРОЦЕДУРА PieSlice

Синтаксис: Procedure PieSlice(X, Y: Integer; StartAngle, EndAngle, Radius: Word)

Описание: выводит сектор круга.

В каком режиме используется: в графическом.

ПРОЦЕДУРА PUTIMAGE

Синтаксис: Procedure PutImage(X, Y: Integer;
Var BitMap; BitBlt: Word)

Описание: выводит спрайт из памяти на экран.

В каком режиме используется: в графическом.

ПРОЦЕДУРА PUTPIXEL

Синтаксис: Procedure PutPixel(X, Y: Integer;
Pixel: Word)

Описание: выводит точку.

В каком режиме используется: в графическом.

ПРОЦЕДУРА RECTANGLE

Синтаксис: Procedure Rectangle(x1, y1, x2,
y2: Integer)

Описание: выводит незакрашенный прямоугольник.

В каком режиме используется: в графическом.

ФУНКЦИЯ REGISTERBGIDRIVER

Синтаксис: Function RegisterBGIdriver(Driver:
Pointer): Integer

Описание: регистрирует графический драйвер, добавленный при построении программы к исполняемому файлу.

В каком режиме используется: в текстовом, графическом.

ФУНКЦИЯ REGISTERBGIFONT

Синтаксис: Function RegisterBGIfont(Font: Pointer):
Integer

Описание: регистрирует шрифт, добавленный при построении программы к исполняемому файлу.

В каком режиме используется: в графическом.

ПРОЦЕДУРА **RESTORECrtMODE**

Синтаксис: Procedure RestoreCrtMode

Описание: восстанавливает последний текстовый режим.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА **SECTOR**

Синтаксис: Procedure Sector(x, y: Integer; StAngle, EndAngle, XRadius, YRadius: Word)

Описание: выводит сектор эллипса.

В каком режиме используется: в графическом.

ПРОЦЕДУРА **SETACTIVEPAGE**

Синтаксис: Procedure SetActivePage(Page: Word)

Описание: устанавливает заданную страницу видеопамати активной, то есть принимающей вывод информации.

В каком режиме используется: в графическом.

ПРОЦЕДУРА **SETALLPALETTE**

Синтаксис: Procedure SetAllPalette(Var Palette)

Описание: устанавливает палитру цветов, заданную параметром.

В каком режиме используется: в графическом.

ФУНКЦИЯ **SETASPECTRATIO**

Синтаксис: Function SetAspectRatio(Xasp,
 Yasp: Word): Word

Описание: устанавливает характеристики отношения вертикального и горизонтального размера экрана (в точках) для избежания эллиптической формы окружностей.

В каком режиме используется: в графическом.

ПРОЦЕДУРА SETBKCOLOR

Синтаксис: Procedure SetBkColor(ColorNum: Word)

Описание: устанавливает цвет фона.

В каком режиме используется: в графическом.

ПРОЦЕДУРА SETCOLOR

Синтаксис: Procedure SetColor(Color: Word)

Описание: устанавливает текущий цвет вывода информации.

В каком режиме используется: в графическом.

ПРОЦЕДУРА SETFILLPATTERN

Синтаксис: Procedure SetFillPattern(Pattern: FillPatternType; Color: Word)

Описание: устанавливает шаблон заливки.

В каком режиме используется: в графическом.

ПРОЦЕДУРА SETFILLSTYLE

Синтаксис: Procedure SetFillStyle(Pattern:
Word; Color: Word)

Описание: устанавливает характеристики заливки.

В каком режиме используется: в графическом.

ПРОЦЕДУРА SETGRAPHBUFSIZE

Синтаксис: Procedure SetGraphBufSize (BufSize: Word)

Описание: изменяет размер буфера, используемого при заливке области процедурой FloodFill.

В каком режиме используется: в графическом.

ПРОЦЕДУРА SETGRAPHMODE

Синтаксис: Procedure SetGraphMode (Mode: Integer)

Описание: устанавливает заданный графический режим для активного графического драйвера.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА SETLINESTYLE

Синтаксис: Procedure SetLineStyle (LineStyle: Word; Pattern: Word; Thickness: Word)

Описание: устанавливает стиль обводки.

В каком режиме используется: в графическом.

ПРОЦЕДУРА SETPALETTE

Синтаксис: Procedure SetPalette (ColorNum: Word; Color: Shortint)

Описание: устанавливает один цвет палитры на основе параметра типа ShortInt.

В каком режиме используется: в графическом.

ПРОЦЕДУРА SETRGBPALETTE

Синтаксис: Procedure SetRGBPalette (ColorNum, RedValue, GreenValue, BlueValue: Integer)

Описание: устанавливает один цвет палитры на основе цветовых составляющих.

В каком режиме используется: в графическом.

ПРОЦЕДУРА SETTEXTJUSTIFY

Синтаксис: Procedure SetTextJustify(Horiz, Vert: Word)

Описание: устанавливает текущее выравнивание текста.

В каком режиме используется: в графическом.

ПРОЦЕДУРА SETTEXTSTYLE

Синтаксис: Procedure SetTextStyle(Font, Direction: Word; CharSize: Word)

Описание: устанавливает характеристики вывода текста.

В каком режиме используется: в графическом.

ПРОЦЕДУРА SETUSERCHARSIZE

Синтаксис: Procedure SetUserCharSize(MultX, DivX, MultY, DivY: Word)

Описание: изменяет стандартные размеры символов текущего шрифта.

В каком режиме используется: в графическом.

ПРОЦЕДУРА SETVIEWPORT

Синтаксис: Procedure SetViewport(x1, y1, x2, y2: Integer; Clip: Boolean)

Описание: устанавливает область отсечения, используемую для предотвращения вывода графической информации за пределы заданного прямоугольника — виртуального окна.

В каком режиме используется: в графическом.

ПРОЦЕДУРА **SETVISUALPAGE**

Синтаксис: Procedure SetVisualPage (Page: Word)

Описание: делает заданную страницу видеопамати видимой — отображаемой на экране.

В каком режиме используется: в графическом.

ПРОЦЕДУРА **SETWRITEMODE**

Синтаксис: Procedure SetWriteMode (WriteMode: Integer)

Описание: устанавливает режим прорисовки линий, выводимых процедурами DrawPoly, Line, LineRel, LineTo и Rectangle на основе побитовых операций с точками, аналогичными методам вывода спрайта на экран.

В каком режиме используется: в графическом.

ФУНКЦИЯ **TEXTHEIGHT**

Синтаксис: Function TextHeight (TextString: String): Word

Описание: возвращает высоту заданной строки для текущего шрифта.

В каком режиме используется: в графическом.

ФУНКЦИЯ **TEXTWIDTH**

Синтаксис: Function TextWidth (TextString: String): Word

Описание: возвращает ширину заданной строки для текущего шрифта.

В каком режиме используется: в графическом.

ПОДПРОГРАММЫ МОДУЛЯ SYSTEM

Модуль `System` содержит разнородные по своему назначению подпрограммы, которые можно разбить на следующие группы:

- Работы с порядковыми типами данных
- Преобразования типов
- Арифметические
- Обработки строк
- Контроля выполнения программы
- Ввода / вывода
- Работы с текстовыми файлами
- Работы с нетипизированными файлами
- Работы с динамической памятью
- Работы с указателями и адресами
- Другие, не вошедшие в предыдущие группы

Данный модуль существенно отличается от всех остальных. Во-первых, его не нужно подключать в разделе подключения библиотек `Uses`, это происходит автоматически. Во-вторых, подпрограммы, включенные в модуль, содержат нетипичные для Turbo Pascal описания, в которые могут входить:

- Необязательные параметры;
- Отсутствие конкретного типа параметра;
- Отсутствие конкретного типа возвращаемого функцией значения.

Необязательные параметры при описании заключаются в квадратные скобки:

```
Function Concat(s1 [, s2, ..., sn]: String): String
```

Описание такого рода означает, что при вызове функции `Concat` (возвращает сумму заданных строк) должен обязательно указываться один параметр типа `String` и еще несколько параметров такого же типа по необходимости.

Если в качестве параметра подпрограммы могут указываться значения различного типа, то в описании такого параметра тип не указывается вообще:

```
Procedure Val(S; Var V; Var Code: Integer)
```

Описание такого рода означает, что при вызове процедуры Val в качестве параметра S может указываться переменная или константа любого типа (данная процедура возвращает строковое представление заданного параметра), а в качестве параметра V — переменная любого типа. На самом деле это немного не так, а конкретные допустимые типы параметром можно найти в разделе справочной системы, посвященной подпрограмме. Например, про параметр S процедуры Val указано: "string-type variable; must be a sequence of characters that form a signed whole number" — переменная строкового типа (то есть String, Char и производные от них), должна быть последовательностью символов, полностью представляющих собой знаковое число.

Некоторые функции возвращают значение, тип которого соответствует типу значения, заданного в качестве ее параметра:

```
Function Sqr(X): <Такой же, как параметр>
```

Это значит, что при параметре типа Integer, значение функции (данная функция возвращает значение параметра в квадрате) также будет иметь тип Integer, а при параметре типа Real, значение функции будет иметь тип Real.

Разные расширенные возможности подпрограмм модуля System могут встречаться одновременно в одном описании:

```
Procedure Inc(Var X [ ; N: Longint ] );
```

В приведенном описании встречается необязательный параметр (параметр N типа LongInt), а параметр-переменная X может иметь любой тип.

ПОДПРОГРАММЫ РАБОТЫ С ПОРЯДКОВЫМИ ТИПАМИ ДАННЫХ

Порядковые типы данных — это простые и сложные типы, все значения области значений которых представляют собой некоторый дискретный ряд. Например, Integer и Byte — порядковые типы, так как их области значений перечислимы (–32768,

–32767, ..., 0, 1, ... 32766, 32767), перечислимые типы также являются порядковыми, а тип `Real` к порядковым не относится, так как множество его значений не является дискретным (количество различных значений, которые могут принимать переменные данного типа в существенной степени определяется сопроцессором).

ПРОЦЕДУРА `Dec`

Синтаксис: `Procedure Dec (Var X [ ; N: Longint ] )`

Описание: присваивает переменной, заданной в качестве, предыдущее значение из области допустимых значений для ее типа. Для целочисленных переменных данное действие аналогично уменьшению значения переменной на единицу. Целочисленный параметр `N` позволяет выполнять смещение значения более чем на единицу.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА `Inc`

Синтаксис: `Procedure Inc (Var X [ ; N: Longint ] )`

Описание: Присваивает переменной, заданной в качестве, следующее значение из области допустимых значений для ее типа. Для целочисленных переменных данное действие аналогично увеличению значения переменной на единицу. Целочисленный параметр `N` позволяет выполнять смещение значения более чем на единицу.

В каком режиме используется: в текстовом, графическом.

Функция `Odd`

Синтаксис: `Function Odd (X: Longint): Boolean`

Описание: проверяет, является ли значение параметра нечетным числом.

В каком режиме используется: в текстовом, графическом.

Функция PRED

Синтаксис: Function Pred(X): < Same type as parameter >

Описание: проверяет, является ли значение параметра четным числом.

В каком режиме используется: в текстовом, графическом.

Функция Succ

Синтаксис: Function Succ(X): (Same type as parameter)

Описание: возвращает следующее значение из области допустимых значений для типа переменной, заданной в качестве параметра. Для целочисленных переменных данное действие аналогично увеличению значения переменной на единицу.

В каком режиме используется: в текстовом, графическом.

ПОДПРОГРАММЫ ПРЕОБРАЗОВАНИЯ ТИПОВ

Функция CHR

Синтаксис: Function Chr(X: Byte): Char

Описание: возвращает символ, номер которого в таблице символов равен значению, заданному параметру.

В каком режиме используется: в текстовом, графическом.

Функция ORD

Синтаксис: Function Ord(X): Longint

Описание: возвращает порядковый номер значения заданного параметра в области возможных значений для его типа.

В каком режиме используется: в текстовом, графическом.

Функция ROUND

Синтаксис: Function Round(X: Real): Longint

Описание: возвращает целое значение ближайшее к значению параметра.

В каком режиме используется: в текстовом, графическом.

Функция TRUNC

Синтаксис: Function Trunc(X: Real): Longint

Описание: возвращает целое значение путем отбрасывания дробной части значения параметра.

В каком режиме используется: в текстовом, графическом.

АРИФМЕТИЧЕСКИЕ ПОДПРОГРАММЫ**Функция** ABS

Синтаксис: Function Abs(X): <Такой же, как тип параметра>

Описание: возвращает модуль значения параметра.

В каком режиме используется: в текстовом, графическом.

Функция ARCTAN

Синтаксис: Function ArcTan(X: Real): Real

Описание: возвращает арктангенс значения параметра.

В каком режиме используется: в текстовом, графическом.

Функция COS

Синтаксис: Function Cos(X: Real): Real

Описание: возвращает косинус значения параметра.

В каком режиме используется: в текстовом, графическом.

Функция Exp

Синтаксис: Function Exp(X: Real): Real

Описание: возвращает экспоненту (число e в степени, равной значению параметра — e^x) значения параметра.

В каком режиме используется: в текстовом, графическом.

Функция Frac

Синтаксис: Function Frac(X: Real): Real

Описание: возвращает дробную часть значения параметра.

В каком режиме используется: в текстовом, графическом.

Функция Int

Синтаксис: Function Int(X: Real): Real

Описание: возвращает целую часть значения параметра (округляет значение параметра в сторону нуля).

В каком режиме используется: в текстовом, графическом.

Функция Ln

Синтаксис: Function Ln(X: Real): Real

Описание: возвращает значение натурального логарифма значения параметра.

В каком режиме используется: в текстовом, графическом.

Функция Pi

Синтаксис: Function Pi: Real

Описание: возвращает число "Пи" (3.1415926535897932385).

В каком режиме используется: в текстовом, графическом.

Функция SIN

Синтаксис: Function Sin(X: Real): Real

Описание: возвращает синус значения параметра.

В каком режиме используется: в текстовом, графическом.

Функция SQR

Синтаксис: Function Sqr(X): <Такой же, как тип параметра>

Описание: возвращает квадрат значения параметра.

В каком режиме используется: в текстовом, графическом.

Функция SQRT

Синтаксис: Function Sqrt(X: Real): Real

Описание: возвращает квадратный корень значения параметра.

В каком режиме используется: в текстовом, графическом.

ПОДПРОГРАММЫ ОБРАБОТКИ СТРОК**Функция CONCAT**

Синтаксис: Function Concat(s1 [, s2, ..., sn]: String): String

Описание: возвращает сумму строк, заданных в качестве параметров (операция конкатенации).

В каком режиме используется: в текстовом, графическом.

Функция COPY

Синтаксис: Function Copy(S: String;
Index: Integer; Count: Integer): String

Описание: возвращает подстроку заданной строки, заключенную между двумя символами с индексами, заданными параметрами.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА DELETE

Синтаксис: Procedure Delete (Var S: String;
Index: Integer; Count: Integer)

Описание: удаляет подстроку заданной строки, заключенную между двумя символами с индексами, заданными параметрами.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА INSERT

Синтаксис: Procedure Insert (Source: String;
Var S: String; Index: Integer)

Описание: вставляет заданную параметром Source строку в строку, заданную параметром S, начиная с заданного индекса.

В каком режиме используется: в текстовом, графическом.

ФУНКЦИЯ LENGTH

Синтаксис: Function Length (S: String): Integer

Описание: возвращает размер (количество символов) заданной строки.

В каком режиме используется: в текстовом, графическом.

ФУНКЦИЯ Pos

Синтаксис: Function Pos (Substr: String;
S: String): Byte

Описание: возвращает номер символа, который начинает первое вхождение подстроки, заданной параметром Substr, в строке S.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА STR

Синтаксис: Procedure Str (X [: Width [: Decimals
]); Var S: String)

Описание: заносит в заданную строку символьное представление параметра `X`. В качестве необязательных параметров можно указать формат представления числа.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА `VAL`

Синтаксис: `Procedure Val (S; Var V;
Var Code: Integer)`

Описание: заносит в переменную, заданную в качестве параметра `V` числовое значение, задаваемое параметром `S` любого строкового типа. Результат операции (0, если преобразование успешно) помещается в переменную, заданную в качестве параметра `Code`.

В каком режиме используется: в текстовом, графическом.

ПОДПРОГРАММЫ КОНТРОЛЯ ВЫПОЛНЕНИЯ ПРОГРАММЫ

ПРОЦЕДУРА `EXIT`

Синтаксис: `Procedure Exit`

Описание: досрочно прекращает выполнение текущей подпрограммы, в том числе и основной части программы, если в данный момент выполняется она.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА `HALT`

Синтаксис: `Procedure Halt [ ( Exitcode: Word ) ]`

Описание: досрочно прекращает выполнение программы. Необязательный параметр указывает код завершения программы, который может быть проанализирован вызывающим процессом.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА RUNERROR

Синтаксис: Procedure RunError [(Errorcode: Byte)]

Описание: останавливает выполнение программы. Необязательный параметр определяет номер ошибки, которая будет выведена после завершения программы.

В каком режиме используется: в текстовом, графическом.

ПОДПРОГРАММЫ ВВОДА / ВЫВОДА

ПРОЦЕДУРА ASSIGN

Синтаксис: Procedure Assign(Var f; S: String)

Описание: сопоставляет файловую переменную любого типа файлу на диске с заданным именем.

В каком режиме используется: в текстовом, графическом.

ФУНКЦИЯ ASSIGNED

Синтаксис: Function Assigned(Var P): Boolean

Описание: определяет равенство параметра значению nil.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА CHDIR

Синтаксис: Procedure ChDir(S: String)

Описание: изменяет текущий каталог.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА CLOSE

Синтаксис: Procedure Close(Var F)

Описание: закрывает заданный файл.

В каком режиме используется: в текстовом, графическом.

Функция Eof

Синтаксис для нетипизированных файлов: `Function Eof(Var F): Boolean`

Синтаксис для текстовых файлов: `Function Eof [ (Var F: Text) ]: Boolean`

Описание: определяет, достигнут ли конец заданного файла.

В каком режиме используется: в текстовом, графическом.

Процедура Erase

Синтаксис: `Procedure Erase(Var F)`

Описание: удаляет заданный файл.

В каком режиме используется: в текстовом, графическом.

Функция FilePos

Синтаксис: `Function FilePos(Var F): Longint`

Описание: определяет позицию текущего указателя в заданном файле.

В каком режиме используется: в текстовом, графическом.

Функция FileSize

Синтаксис: `Function FileSize(Var F): Longint`

Описание: определяет размер заданного файла.

В каком режиме используется: в текстовом, графическом.

Процедура GetDir

Синтаксис: `Procedure GetDir(D: Byte; Var S: String)`

Описание: возвращает название текущего каталога для диска, заданного параметром D, (0 для текущего диска, 1 для диска A:, 2 для диска B:, 3 для диска C:).

В каком режиме используется: в текстовом, графическом.

Функция IOResult

Синтаксис: Function IOResult: Integer

Описание: возвращает код ошибки последней операции ввода/вывода.

В каком режиме используется: в текстовом, графическом.

Процедура MkDir

Синтаксис: Procedure MkDir(S: String)

Описание: создает каталог.

В каком режиме используется: в текстовом, графическом.

Процедура Rename

Синтаксис: Procedure Rename(Var F; Newname)

Описание: переименовывает файл.

В каком режиме используется: в текстовом, графическом.

Процедура Reset

Синтаксис: Procedure Reset(Var F [: File; Recsize: Word])

Описание: открывает файл для чтения.

В каком режиме используется: в текстовом, графическом.

Процедура Rewrite

Синтаксис: Procedure Rewrite (Var F: File; Rec-size: Word])

Описание: открывает файл для записи.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА RmDir

Синтаксис: Procedure RmDir (S: String)

Описание: удаляет пустой каталог.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА Seek

Синтаксис: Procedure Seek (Var F; N: Longint)

Описание: изменяет положение текущего указателя в заданном файле.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА Truncate

Синтаксис: Procedure Truncate (Var F)

Описание: удаляет часть заданного файла, находящуюся за позицией текущего указателя.

В каком режиме используется: в текстовом, графическом.

ПОДПРОГРАММЫ РАБОТЫ С ТЕКСТОВЫМИ ФАЙЛАМИ

ПРОЦЕДУРА Append

Синтаксис: Procedure Append (Var f: Text)

Описание: открывает текстовый файл для добавления информации.

В каком режиме используется: в текстовом, графическом.

Функция EOLN

Синтаксис: Function Eoln [(Var F: Text)]: Boolean

Описание: определяет, достигнут ли в заданном файле конец строки.

В каком режиме используется: в текстовом, графическом.

Процедура FLUSH

Синтаксис: Procedure Flush(Var F: Text)

Описание: записывает в файл буфер из памяти.

В каком режиме используется: в текстовом, графическом.

Процедура READ

Синтаксис для типизированных файлов: Procedure Read(F, V1 [, V2, ..., Vn])

Синтаксис для текстовых файлов: Procedure Read([Var F: Text;] V1 [, V2, ..., Vn])

Описание: читает информацию. Необязательный параметр указывает, из какого файла следует считывать информацию.

В каком режиме используется: в текстовом, графическом.

Процедура READLN

Синтаксис: Procedure Readln([[Var F: Text;] V1 [, V2, ..., Vn]])

Описание: читает информацию. Необязательный параметр указывает, из какого файла следует считывать информацию.

В каком режиме используется: в текстовом, графическом.

Функция SEEKEOF

Синтаксис: Function SeekEof [(Var F: Text)]: Boolean

Описание: определяет, достигнут ли конец заданного текстового файла.

В каком режиме используется: в текстовом, графическом.

Функция SEEKEOLN

Синтаксис: Function SeekEoln [(Var F: Text)]: Boolean

Описание: определяет, достигнут ли конец строки в заданном текстовом файле.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА SETTEXTBUF

Синтаксис: Procedure SetTextBuf(Var F: Text; Var Buf [; Size: Word])

Описание: устанавливает буфер для операций чтения/записи. Необязательный параметр определяет размер буфера.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА WRITE

Синтаксис для типизированных файлов: Procedure Write(F, V1 [, V2, ..., Vn])

Синтаксис для текстовых файлов: Procedure Write([Var F: Text;] P1 [, P2, ..., Pn])

Описание: записывает информацию. Необязательный параметр указывает, в какой файл следует записывать информацию.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА WRITELN

Синтаксис: Procedure Writeln([Var F: Text;] P1
[, P2, ..., Pn])

Описание: записывает информацию. Необязательный параметр указывает, в какой файл следует записывать информацию.

В каком режиме используется: в текстовом, графическом.

ПОДПРОГРАММЫ РАБОТЫ С НЕТИПИЗИРОВАННЫМИ ФАЙЛАМИ

ПРОЦЕДУРА BlockRead

Синтаксис: Procedure BlockRead(Var F: File; Var
Buf; Count: Word [; Var Result: Word])

Описание: записывает информацию в заданный файл.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА BlockWrite

Синтаксис: Procedure BlockWrite(Var f: File; Var
Buf; Count: Word [; Var Result: Word])

Описание: записывает информацию в заданный файл.

В каком режиме используется: в текстовом, графическом.

ПОДПРОГРАММЫ РАБОТЫ С ДИНАМИЧЕСКОЙ ПАМЯТЬЮ

ПРОЦЕДУРА Dispose

Синтаксис: Procedure Dispose(Var P: Pointer [,
Destructor])

Описание: освобождает память, на которую указывает параметр. Необязательный параметр указывает деструктор, если память выделена под экземпляр объекта.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА FREEMEM

Синтаксис: Procedure FreeMem(Var P: Pointer; Size: Word)

Описание: освобождает память заданного размера, на которую указывает параметр.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА GETMEM

Синтаксис: Procedure GetMem(Var P: Pointer; Size: Word)

Описание: выделяет память заданного размера под заданный указатель.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА NEW

Синтаксис: Procedure New(Var P: Pointer [; Init: Constructor])

Описание: выделяет память под заданный указатель. Необязательный параметр указывает конструктор, если память выделяется под экземпляр объекта.

В каком режиме используется: в текстовом, графическом.

ФУНКЦИЯ MAXAVAIL

Синтаксис: Function MaxAvail: Longint

Описание: возвращает размер максимального блока динамической памяти, который может быть выделен под указатель (реально свободная память в контексте необходимого размера).

В каком режиме используется: в текстовом, графическом.

ФУНКЦИЯ MEMAVAIL

Синтаксис: Function MemAvail: Longint

Описание: возвращает размер свободной динамической памяти (теоретически свободная память).

В каком режиме используется: в текстовом, графическом.

ПОДПРОГРАММЫ РАБОТЫ С УКАЗАТЕЛЯМИ И АДРЕСАМИ

Функция ADDR

Синтаксис: Function Addr(X): Pointer

Описание: возвращает адрес переменной или подпрограммы, заданной в качестве параметра.

В каком режиме используется: в текстовом, графическом.

Функция CSEG

Синтаксис: Function CSeg: Word

Описание: возвращает текущее значение регистра CS.

В каком режиме используется: в текстовом, графическом.

Функция DSEG

Синтаксис: Function DSeg: Word

Описание: возвращает текущее значение регистра DS.

В каком режиме используется: в текстовом, графическом.

Функция OFS

Синтаксис: Function OfS(X): Word

Описание: возвращает смещение заданного параметра в сегменте.

В каком режиме используется: в текстовом, графическом.

Функция PTR

Синтаксис: Function Ptr(Seg, Ofs: Word): Pointer

Описание: возвращает указатель на адрес памяти, заданный сегментом и смещением.

В каком режиме используется: в текстовом, графическом.

Функция SEG

Синтаксис: Function Seg(X): Word

Описание: возвращает сегмент, в котором находится заданный параметр.

В каком режиме используется: в текстовом, графическом.

Функция SPTR

Синтаксис: Function SPtr: Word

Описание: возвращает текущее значение регистра SP.

В каком режиме используется: в текстовом, графическом.

Функция SSEG

Синтаксис: Function SSeg: Word

Описание: возвращает текущее значение регистра SS.

В каком режиме используется: в текстовом, графическом.

ДРУГИЕ ПОДПРОГРАММЫ**Процедура** EXCLUDE

Синтаксис: Procedure Exclude(Var S: Set Of T; I: T)

Описание: исключает заданный элемент из заданного множества.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА FILLCHAR

Синтаксис: Procedure FillChar(Var X; Count: Word; Value)

Описание: заполняет заданную параметром X область памяти, размером Count. Память заполняется значением параметра Value.

В каком режиме используется: в текстовом, графическом.

Функция HI

Синтаксис: Function Hi(X): Byte

Описание: возвращает старший байт значения параметра.

В каком режиме используется: в текстовом, графическом.

Функция HIGH

Синтаксис: Function High(X)

Описание: возвращает максимальное значение для типа заданного параметра.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА INCLUDE

Синтаксис: Procedure Include(Var S: Set Of T; I: T)

Описание: включает заданный элемент в заданное множество.

В каком режиме используется: в текстовом, графическом.

Функция LO

Синтаксис: Function Lo(X): Byte;

Описание: возвращает младший байт значения параметра.

В каком режиме используется: в текстовом, графическом.

Функция Low

Синтаксис: Function Low(X);

Описание: возвращает минимальное значение для типа заданного параметра.

В каком режиме используется: в текстовом, графическом.

Процедура Move

Синтаксис: Procedure Move(Var Source, Dest;
Count: Word)

Описание: копирует информацию из переменной, заданной параметром Source в переменную, заданную параметром Dest. Размер области копирования задается параметром Count.

В каком режиме используется: в текстовом, графическом.

Функция ParamCount

Синтаксис: Function ParamCount: Word

Описание: возвращает количество параметров, с которыми вызвана программа.

В каком режиме используется: в текстовом, графическом.

Функция ParamStr

Синтаксис: Function ParamStr(Index): String

Описание: возвращает параметр программы с заданным номером.

В каком режиме используется: в текстовом, графическом.

Функция Random

Синтаксис: Function Random [(Range: Word)]: <Такой же, как тип параметра>

Описание: возвращает случайное значение от нуля (включительно) до значения параметра (не включая его). Если параметр не задан, возвращается случайное число типа Real.

В каком режиме используется: в текстовом, графическом.

ПРОЦЕДУРА RANDOMIZE

Синтаксис: Procedure Randomize

Описание: изменяет базу генерации случайных чисел на основе системных часов.

В каком режиме используется: в текстовом, графическом.

ФУНКЦИЯ SIZEOF

Синтаксис: Function SizeOf(Var X): Integer

Описание: возвращает размер памяти, необходимый для хранения переменной заданного типа. В качестве параметра может быть указано название типа данных.

В каком режиме используется: в текстовом, графическом.

ФУНКЦИЯ SWAP

Синтаксис: Function Swap(X): <Такой же, как тип параметра>

Описание: обменивает младший и старший байты параметра значениями.

В каком режиме используется: в текстовом, графическом.

ФУНКЦИЯ UPCASE

Синтаксис: Function UpCase(Ch: Char): Char

Описание: переводит латинские символы заданной строки в верхний регистр.

В каком режиме используется: в текстовом, графическом.

СПИСОК И ОПИСАНИЕ СООБЩЕНИЙ ОБ ОШИБКАХ

СООБЩЕНИЯ ОБ ОШИБКАХ В ТЕКСТЕ ПРОГРАММЫ

- | | | |
|----|------------------------------|--|
| 1 | Out of memory | Недостаточно памяти. |
| 2 | Identifier expected | В данном месте должен находиться какой-либо идентификатор. |
| 3 | Unknown identifier | Неизвестный идентификатор (неопи-
санная константа, переменная, название процедуры и т.д.). |
| 4 | Duplicate identifier | Попытка повторного описания иденти-
фикатора. |
| 5 | Syntax error | Синтаксическая ошибка (использование недо-
пустимых символов). |
| 6 | Error in real constant | Ошибка представления вещест-
венного значения. |
| 7 | Error in integer constant | Ошибка представления цело-
численного значения. |
| 8 | String constant exceeds line | Не найден конец стро-
ковой константы (пропущена закрывающая кавычка). |
| 10 | Unexpected end of file | Отсутствует признак окончания
программы (ключевое слово End с точкой) |
| 11 | Line too long | Слишком длинная строка в программе (макси-
мально допустимая длина строки — 127 символов). |
| 12 | Type identifier expected | Не указан тип идентификатора. |
| 13 | Too many open files | Слишком много открытых файлов. |
| 14 | Invalid file name | Некорректное имя файла. |
| 15 | File not found | Файл не найден. |
| 16 | Disk full | Диск переполнен, запись информации на него
невозможна. |

- 17 Invalid compiler directive Некорректная директива компилятору.
- 18 Too many files Слишком много файлов, которые участвуют в построении программы.
- 19 Undefined type in pointer def Не определен тип переменной-указателя.
- 20 Variable identifier expected Требуется указание имени переменной.
- 21 Error in type некорректное использование типа.
- 22 Structure too large Структура имеет слишком большой размер (превышающий 65520 байт).
- 23 Set base type out of range Базовый тип множества содержит слишком много элементов (более 256)
- 24 File components may not be files or objects В качестве компонентов файловой переменной не могут быть файлы и объекты.
- 25 Invalid string length Некорректная длина строки (строка может иметь длину от 1 до 255)
- 26 Type mismatch Несоответствие типов.
- 27 Invalid subrange base type Недопустимо использовать данный тип для формирования типа-диапазона.
- 28 Lower bound > than upper bound Нижняя граница типа-диапазона больше верхней.
- 29 Ordinal type expected В данном месте должен быть использован порядковый тип данных.
- 30 Integer constant expected В данном месте должна быть использована целочисленная константа.
- 31 Constant expected В данном месте должна быть использована константа.
- 32 Integer or real constant expected В данном месте должна быть использована вещественная или целочисленная константа.

- 33 Pointer Type identifier expected В данном месте должна быть использована переменная-указатель.
- 34 Invalid function result type Неправильный тип возвращаемого функцией значения.
- 35 Label identifier expected В данном месте должна быть использована метка безусловного перехода.
- 36 BEGIN expected В данном месте должно быть использовано ключевое слово begin.
- 37 END expected В данном месте должно быть использовано ключевое слово end.
- 38 Integer expression expected В данном месте должно быть использовано выражение с целочисленным результатом.
- 39 Ordinal expression expected В данном месте должно быть использовано выражение с результатом порядкового типа.
- 40 Boolean expression expected В данном месте должно быть использовано выражение с результатом типа Boolean.
- 41 Operand types do not match Использование операндов недопустимого типа.
- 42 Error in expression Ошибка в выражении.
- 43 Illegal assignment Некорректное присвоение.
- 44 Field identifier expected В данном месте должен быть использован идентификатор поля.
- 45 Object file too large Объектный файл слишком велик (превышает 64Кб).
- 46 Undefined EXTERN Не найдена указанная внешняя процедура.
- 47 Invalid object file record Некорректный формат объектного файла.
- 48 Code segment too large (превышает 65520 байт) Сегмент кода слишком велик
- 49 Data segment too large (превышает 65520 байт) Сегмент данных слишком велик

- 50 DO expected В данном месте должно быть использовано ключевое слово Do.
- 51 Invalid PUBLIC definition Некорректное Public-определение.
- 52 Invalid EXTRN definition Некорректное Extern-определение.
- 53 Too many EXTRN definitions Слишком много Extern-определений.
- 54 OF expected В данном месте должно быть использовано ключевое слово Of.
- 55 INTERFACE expected В данном месте должно быть использовано ключевое слово Interface (отсутствует секция интерфейсных описаний).
- 56 Invalid relocatable reference Некорректная перемещаемая ссылка.
- 57 THEN expected В данном месте должно быть использовано ключевое слово Then.
- 58 TO or DOWNT0 expected В данном месте должны быть использованы ключевые слова To или DownTo.
- 59 Undefined forward В описательной секции отсутствует описание подпрограммы, заявленной в интерфейсной секции.
- 61 Invalid typecast неправильное преобразование типов.
- 62 Division by zero Деление на ноль.
- 63 Invalid file type Подпрограмма работы с файлами неприменима для файлов данного типа.
- 64 Cannot read or write variables of this type Невозможно вывести или считать значение переменной данного типа.
- 65 Pointer variable expected В данном месте должна быть использована переменная-указатель.
- 66 String variable expected В данном месте должна быть использована переменная типа String.

- 67 String expression expected В данном месте должно быть использовано выражение типа String.
- 68 Circular unit reference Циклическая ссылка интерфейсных частей модулей друг на друга.
- 69 Unit name mismatch Несоответствие названия модуля, указанного в разделе Unit и имени файла.
- 70 Unit version mismatch Неправильная версия файла модуля.
- 71 Internal stack overflow Внутренняя ошибка переполнения стека.
- 72 Unit file format error Некорректный формат файла модуля.
- 73 Implementation expected Отсутствует указание начала описательной секции модуля.
- 74 Constant and case types don't match Константа, указанная в качестве возможного значения переменной, анализируемой оператором Case, не соответствует типу данной переменной.
- 75 Record or object variable expected В данном месте должна быть использована переменная-запись или переменная-экземпляр объекта.
- 76 Constant out of range Константа выходит за пределы допустимых значений.
- 77 File variable expected В данном месте должна быть использована файловая переменная.
- 78 Pointer expression expected В данном месте должно быть использовано выражение типа Pointer.
- 79 Integer or real expression expected В данном месте должно быть использовано выражение целочисленного или вещественного типа.
- 80 Label not within current block Метка безусловного перехода находится за пределами текущего блока.

- 81 Label already defined Данная метка безусловного перехода уже установлена.
- 82 Undefined label in preceding statement part М е т - ка безусловного перехода, на которую осуществляется переход из предшествующего фрагмента программы, не установлена.
- 83 Invalid @ argument Некорректный аргумент оператора получения адреса @.
- 84 UNIT expected В данном месте должно быть использовано ключевое слово Unit.
- 85 «;» expected В данном месте должен быть использован символ “;”.
- 86 «:» expected В данном месте должен быть использован символ “:”.
- 87 «,» expected В данном месте должен быть использован символ “,”.
- 88 «(» expected В данном месте должен быть использован символ “(”.
- 89 «)» expected В данном месте должен быть использован символ “)”.
- 90 «=» expected В данном месте должен быть использован оператор “=”.
- 91 «:=» expected В данном месте должен быть использован оператор “:=”.
- 92 «[» or “(.” Expected В данном месте должен быть использован символ “[”.
- 93 «]» or “.)” expected В данном месте должен быть использован символ “]”.
- 94 «.» expected В данном месте должен быть использован символ “.”.
- 95 «..» expected В данном месте должен быть использован символ “..”.

- 96 Too many variables Описано слишком много переменных (превышение максимально допустимого размера сегмента данных).
- 97 Invalid FOR control variable Используется некорректный счетчик цикла.
- 98 Integer variable expected В данном месте должна быть использована целочисленная переменная.
- 99 Files types are not allowed here В данном контексте не могут быть использованы файловые типы.
- 100 String length mismatch Несоответствие длин строк.
- 101 Invalid ordering of fields Неправильный порядок полей записи.
- 102 String constant expected В данном месте должна быть использована строковая константа.
- 103 Integer or real variable expected В данном месте должна быть использована целочисленная или вещественная переменная.
- 104 Ordinal variable expected В данном месте должна быть использована переменная порядкового типа.
- 105 INLINE error Некорректное использование оператора Inline.
- 106 Character expression expected В данном месте должно быть использовано выражение типа Char.
- 107 Too many relocation items Слишком много перемещаемых элементов.
- 108 Overflow in arithmetic operation Переполнение при выполнении арифметической операции.
- 109 No enclosing For, While or Repeat statement Обнаружен незакрытый оператор For, While или Repeat.
- 112 CASE constant out of range Константа, используемая в операторе Case, выходит за пределы допустимого множества значений.

- 113 Error in statement Ошибка в операторе.
- 114 Cannot call an interrupt procedure невозможно вызвать обработчик прерывания.
- 116 Must be in 8087 mode to compile Данный фрагмент программы можно откомпилировать только в режиме процессора 8087 (с сопроцессором).
- 117 Target address not found Адрес назначения не найден.
- 118 Include files are not allowed here Подключение файлов недопустимо в данном месте.
- 119 No inherited methods are accessible here Н е т унаследованных методов, доступных в данном месте.
- 121 Invalid qualifier Некорректное использование квалификатора (индекса или оператора разыменовывания).
- 122 Invalid variable reference Некорректная ссылка на переменную.
- 123 Too many symbols Слишком много символов.
- 124 Statement part too large Слишком большой размер фрагмента программы, используемого в операторе.
- 126 Files must be var parameters Файловые переменные должны передаваться как параметры-переменные.
- 127 Too many conditional symbols Слишком много директив условной компиляции.
- 128 Misplaced conditional directive Пропущена необходимая директива условной компиляции.
- 129 ENDIF directive missing Пропущена директива условной компиляции ENDIF.
- 130 Error in initial conditional defines Использование некорректных условных определений.
- 131 Header does not match previous definition Заголовок подпрограммы в описательной части не соответствует ее заголовку в интерфейсной части.

- 133 Cannot evaluate this expression Невозможно вычислить значение данного выражения.
- 134 Expression incorrectly terminated Выражение некорректно заканчивается.
- 135 Invalid format specifier Неправильный спецификатор формата.
- 136 Invalid indirect reference Некорректная косвенная ссылка.
- 137 Structured variables are not allowed here В данном месте недопустимо использовать переменную структурного типа.
- 138 Cannot evaluate without System unit Вычисление невозможно в связи с отсутствием модуля System.
- 139 Cannot access this symbol Нет доступа к данному символу.
- 140 Invalid floating-point operation Некорректная операция с плавающей точкой.
- 141 Cannot compile overlays to memory программа, использующая оверлейные модули должна строиться на диске.
- 142 Pointer or procedural variable expected в данном месте необходимо использовать переменную процедурного или ссылочного типа.
- 143 Invalid procedure or function reference Некорректная ссылка на подпрограмму.
- 144 Cannot overlay this unit Данный модуль не может использоваться в качестве оверлейного.
- 145 Too many nested scopes Слишком большая вложенность.
- 146 File access denied Доступ к файлу запрещен.
- 147 object type expected В данном месте должна использоваться переменная-указатель на экземпляр объекта.

- 148 Local object types are not allowed В локальных разделах описания типов описание недопустимо описание объектов.
- 149 Virtual expected в данном месте следует указать ключевое слово virtual.
- 150 Method identifier expected Не указано имя метода.
- 151 Virtual constructors are not allowed Виртуальные конструкторы недопустимы.
- 152 Constructor identifier expected Не указан конструктор.
- 153 Destructor identifier expected Не указан деструктор.
- 154 Fail only allowed within constructors Обращение к процедуре Fail может осуществляться только из конструктора.
- 155 Invalid combination of opcode and operands Некорректное использование операнда для данной команды.
- 156 Memory reference expected В данном месте требуется указание адреса.
- 157 Cannot add or subtract relocatable symbols Складывать и вычитать перемещаемые символы недопустимо.
- 158 Invalid register combination Некорректная комбинация регистров.
- 159 286/287 instructions are not enabled Набор инструкций процессора 286/287 недоступен.
- 160 Invalid symbol reference Некорректная ссылка на символ.
- 161 Code generation error Ошибка генерации кода.
- 162 ASM expected В данном месте должно быть использовано ключевое слово asm.
- 163 Duplicate dynamic method index Повторное использование индекса динамического метода.
- 164 Duplicate resource identifier Повторное использование идентификатора ресурса.

- 165 Duplicate or invalid export clause Попытка использовать индекс повторно, или индекс выходит за пределы области допустимых значений.
- 166 Procedure or function identifier expected В данном месте должен быть вызов подпрограммы.
- 167 Cannot export this symbol Невозможно экспортировать данный символ.
- 168 Duplicate export name Данное имя уже используется другой экспортируемой подпрограммой.
- 169 Executable file header too large Слишком большой заголовок исполняемого файла.

ОШИБКИ ВРЕМЕНИ ВЫПОЛНЕНИЯ

Динамические ошибки, возникающие в процессе выполнения программы, называются ошибками времени выполнения программы приводят к появлению на экране сообщения вида Runtime error nnn at xxxx:yyyy — Ошибка времени выполнения номер nnn по адресу xxxx:yyyy. Ошибок времени выполнения достаточно много, а мы приведем только те, которые могут встретиться при использовании Turbo Pascal.

- 2 File not found Файл не найден.
- 3 Path not found Путь к файлу или каталогу не найден.
- 4 Too many open files Слишком много открытых файлов.
- 5 File access defined Ошибка при доступе к файлу
- 15 Invalid drive number Недопустимый номер диска.
- 16 Cannot remove current directory Невозможно удалить текущий каталог.
- 17 Cannot rename across drives Попытка переименования файла с указанием другого логического диска.
- 100 Disk read error Ошибка чтения с диска.
- 101 Disk write error Ошибка записи на диск.

- 102 File not assigned Файловая переменная не сопоставлена файлу на диске.
- 103 File not open Файл не открыт.
- 104 File not open for input Файл не открыт для ввода.
- 105 File not open for output Файл не открыт для вывода.
- 106 Invalid numeric format Неверный числовой формат.
- 150 Disk is write protected Диск защищен от записи.
- 151 Unknown unit Неизвестный модуль.
- 152 Drive not ready Дисковод не готов.
- 155 Bad drive request structure length При обращении к диску указана неверная длина структуры.
- 156 Disk seek error Ошибка при операции установки головок на диске.
- 157 Unknown media type Неизвестный тип носителя.
- 159 Printer out of paper Кончилась бумага на принтере.
- 160 Device write fault Ошибка при записи на устройство.
- 161 Device read fault Ошибка при чтении с устройства.
- 162 Hardware failure Сбой аппаратуры.
- 200 Division by zero Деление на ноль.
- 201 Range check error Ошибка при проверке границ.
- 202 Stack overflow error Переполнение стека.
- 203 Heap overflow error Недостаток динамической памяти.
- 204 Invalid pointer operation Некорректная операция с указателем.
- 205 Floating point overflow Переполнение при операции с плавающей точкой.
- 206 Floating point underflow Потеря порядка при операции с плавающей точкой.

- 207 Invalid floating point operation Недопустимая операция с плавающей точкой.
- 209 Overlay file read error Ошибка чтения оверлейного файла.
- 210 Object not initialized Не инициализирован экземпляр объекта.
- 211 Call to abstract- method Вызов абстрактного правила.
- 212 Stream registration error Ошибка в регистрируемом потоке.
- 213 Collection index out of range Индекс коллекции выходит за границы диапазона.
- 214 Collection overflow error Переполнение коллекции.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Сухарев М. Turbo Pascal 7.0. Теория и практика программирования - 3 изд., СПб.: Наука и Техника, 2007. - 544 с.
2. Меженный О.А. Turbo Pascal. Краткое руководство. - М.: Издательский дом "Вильямс", 2006. - 240 с.: ил.

Группа подготовки издания:

Зав. редакцией компьютерной литературы: *М.В. Финков*

Редактор: *М.А. Финкова*

Корректоры: *Е.Е. Кириллов, Н.Б. Сиразитдинова*

ООО «Наука и Техника»

198097, Санкт-Петербург, ул. Маршала Говорова, 29

Подписано в печать 17.08.2006. Формат 60×88 1/16.

Бумага газетная. Печать офсетная. Объем 14 п. л.

Тираж 8 000 экз. Заказ № 806

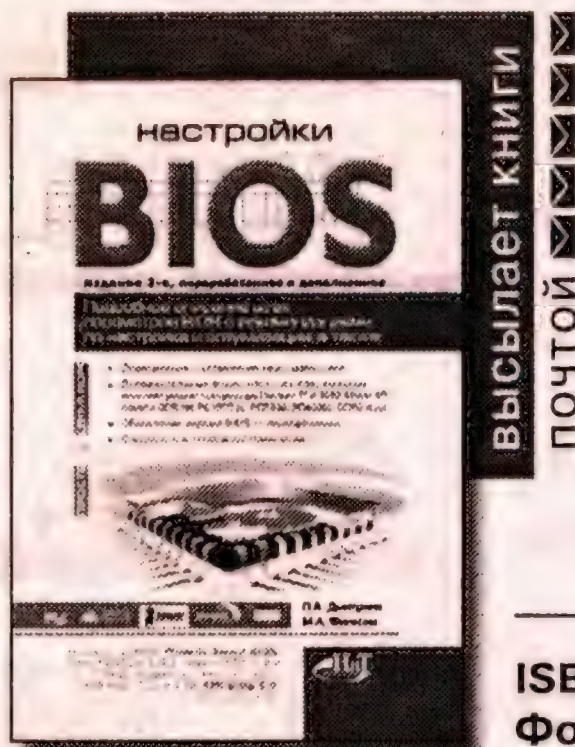
Отпечатано с готовых диапозитивов в ОАО «Техническая книга»

190005, Санкт-Петербург, Измайловский пр., 29

Издательство «Наука и Техника»

П. А. Дмитриев
М. А. Финкова

Настройки BIOS



ISBN: 5-94387-191-8

Формат: 60x88/₁₆

Объем: 288 с.: ил.

Издание 2-е, переработанное и дополненное

В данной книге приведено подробное описание BIOS и его параметров. При этом представлены различные версии BIOS от всех известных производителей, включая самые современные. Рассмотрены все настройки BIOS и даны рекомендации по выбору нужных значений. Приводятся результаты тестирования.

Из этой книги вы узнаете, как повысить производительность компьютера, оптимизировать работу его отдельных систем (процессора, оперативной памяти и т.д.), научитесь решать возникшие проблемы. Вы узнаете о специфических опциях как неновых, так и самых современных устройств (процессоры Pentium от I до IV, AMD от K5 до Athlon XP, чипсеты от Intel, SIS, VIA и т.д.). Целая глава посвящена описанию ошибок и путям их исправления, а завершает книгу подробное описание перепрошивки BIOS и удобный алфавитный указатель.

серия

ПРОСТО О СЛОЖНОМ

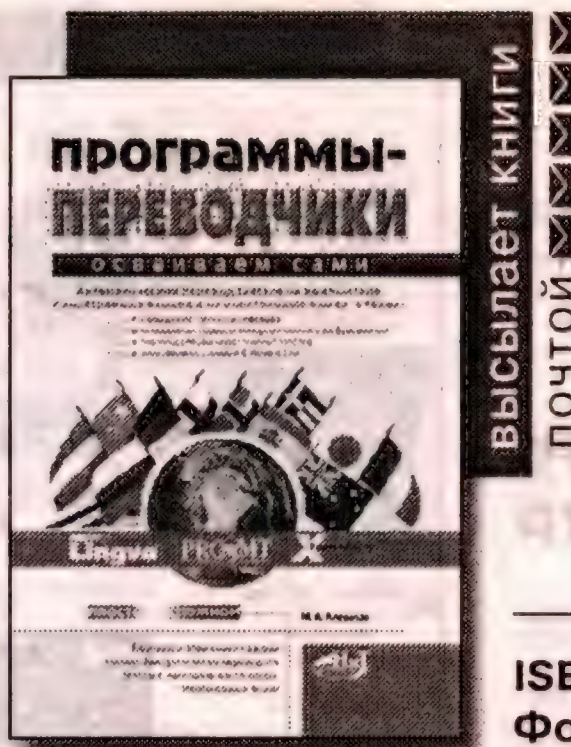
серия

Издательство «Наука и Техника»

М. А. Алешков

Программы-переводчики

Осваиваем сами



ISBN: 5-94387-106-3

Формат: 60x88/₁₆

Объем: 144 с.: ил.

Современные программы-переводчики, устанавливаемые на компьютер, позволяют в автоматическом режиме переводить тексты с иностранного языка и на иностранный язык. Благодаря им, даже не зная иностранного языка, вы сможете быстро и оперативно выполнить перевод. Однако, чтобы перевод был более-менее качественным, необходимо должным образом подготовить текст, а также выполнить определенные настройки программы-переводчика.

В книге вы найдете подробное описание того, что и как нужно делать, чтобы компьютер перевел вам текст. Все расписано буквально по шагам, простым и понятным языком. Поэтому с этой книгой, даже если вы начинающий пользователь ПК, вы без труда сможете выполнять на компьютере переводы.

серия

ПРОСТО О СЛОЖНОМ

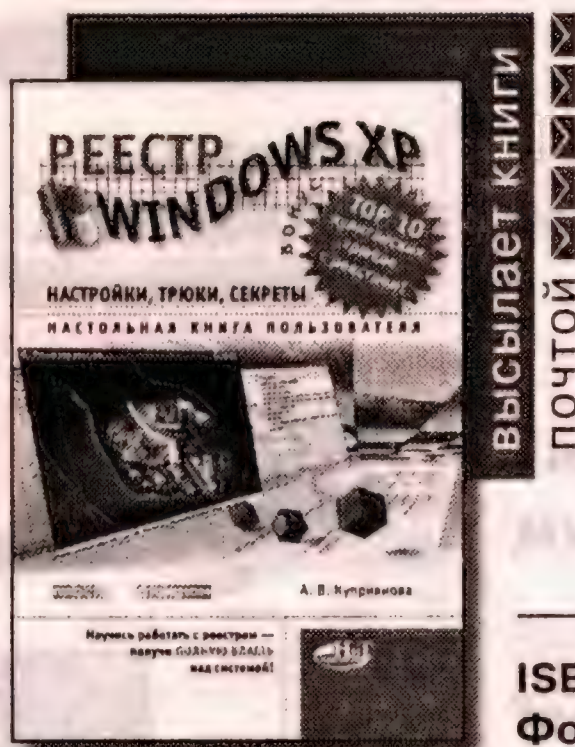
серия

Издательство «Наука и Техника»

А. В. Куприянова

Реестр Windows XP

Настройки, трюки,
секреты.
Настольная книга
пользователя



ISBN: 5-94387-261-2
Формат: 60x88/₁₆
Объем: 192 с.: ил.

Реестр — это база данных, в которой хранится информация обо всех настройках и параметрах работы Windows XP, а также конфигурация всех установленных в системе приложений. С помощью реестра можно делать с системой все что угодно, даже то, что невозможно сделать через стандартные средства Windows XP и ее диалоговые окна.

Эта книга научит вас с помощью реестра проделывать всевозможные трюки с Windows XP, решать проблемы в ее работе, активизировать скрытые механизмы системы и многое другое. Большинство книг по данной теме является излишне теоретизированным и ориентировано сугубо на специалистов. Данная же книга рассчитана на реальные пользовательские задачи, интересы и проблемы. Написана доступным языком с разъяснением всех сопутствующих понятий.

Предназначена самому широкому кругу читателей.

серия

просто о сложном

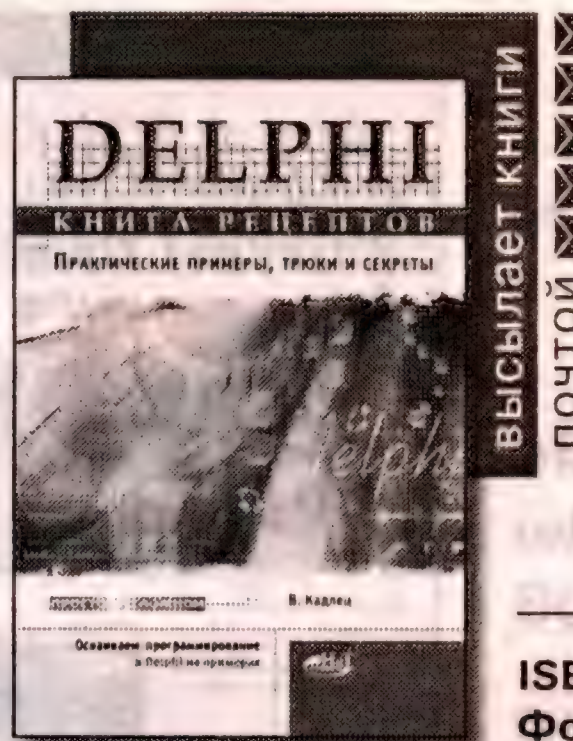
серия

Издательство «Наука и Техника»

В. Кадлец

Delphi: книга рецептов

Практические
примеры, трюки
и секреты



ISBN: 5-94387-269-8

Формат: 70x100/16

Объем: 384 с.: ил.

Данная книга представляет собой великолепное практическое руководство по программированию в Delphi. Каждая глава посвящена отдельному кругу задач и содержит подробное изложение средств, которыми располагает Delphi для решения этих задач. При этом каждое описание сопровождается конкретными примерами. В книге вы найдете готовые решения (программы) из таких разных областей программирования, как разработка пользовательского интерфейса, работа с буфером обмена, системным реестром и сообщениями Windows, динамические библиотеки и параллельное программирование, обмен данными между приложениями и программирование для Интернета, работа с базами данных и подключение к приложению справочной системы, а также многое другое.

Книга предназначена как новичкам, желающим изучить Delphi, так и более опытным программистам. Написана простым и доступным языком, код каждого примера сопровождается подробными комментариями.

серия

ПРОСТО О СЛОЖНОМ

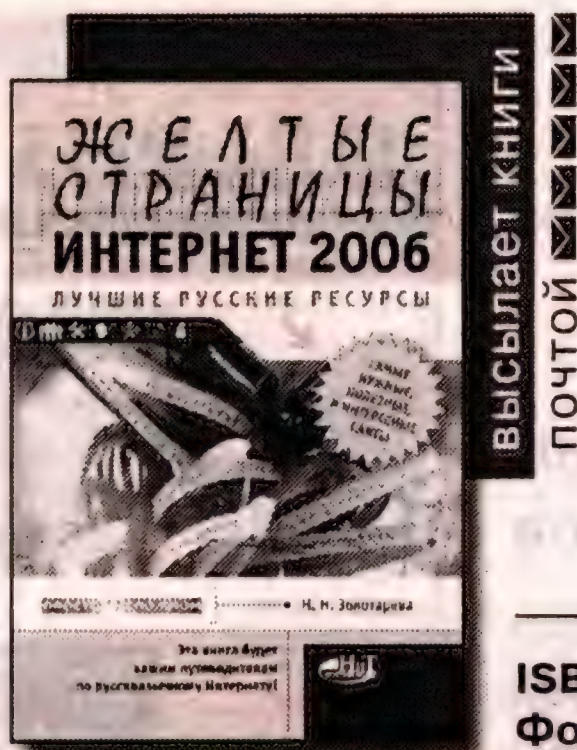
серия

Издательство «Наука и Техника»

Н. Н. Золотарева

Желтые страницы Интернет 2006

Лучшие русские
ресурсы



ISBN: 5-94387-284-1

Формат: 60x88/₁₆

Объем: 368 с.: ил.

В этой книге собраны ссылки на несколько тысяч самых нужных, полезных, популярных и интересных русскоязычных сайтов по самым различным тематикам. Каждая ссылка сопровождается кратким описанием, что позволяет уже по тексту книги ориентироваться среди сайтов.

На поиски и отбор сайтов было потрачено огромное количество времени, которое не надо будет тратить вам. С данной книгой вы всегда сможете быстро найти нужные сайты в Интернете по интересующей вас теме.

Книга предназначена широкому кругу читателей. Она будет вашим путеводителем по русскоязычному Интернету!

серия

ПРОСТО О СЛОЖНОМ

серия

Книги ПОЧТОЙ

Издательство «Наука и Техника» принимает заказы на продажу собственной печатной продукции по почте наложенным платежом.

Оплата производится на почте при получении книг, для организаций возможна оплата по безналичному расчету после выставления счета. Для этого Вам необходимо оформить бланк заказа и отправить его нам.

Для жителей России:

193029 Санкт-Петербург, а/я 44,
ООО «Наука и Техника»
тел/факс (812)-567-70-26, 567-70-25
E-mail: nit@mail.wplus.net

Для жителей Украины:

02166 Киев-166, ул. Курчатова, 9/21,
«Наука и Техника»
тел/факс (044)-516-38-66
E-mail: nits@voliacable.com

**Перечень рассылаемых книг размещен на
сайте издательства: www.nit.com.ru**

Жители Украины могут приобрести книги других издательств:

Радиоэлектроника.....	☐
Компьютерные технологии.....	☐
Техника безопасности на предприятиях	☐
Транспорт	☐

Примечание. Заказанные бесплатные каталоги высылаются по электронной почте или вкладываются в посылку с книгами (отдельно письмом каталоги не рассылаются).

ЗАКАЗ НА КАТАЛОГИ

Заполняйте поля аккуратно большими отдельными буквами.

Информация для приобретения книг почтой частными лицами

1. Фамилия, имя, отчество _____
2. Почтовый адрес: индекс _____ страна _____
область _____ город, поселок _____
улица _____ дом _____ корпус _____ кв. _____
телефон (_____) _____
адрес электронной почты (если он у Вас есть) : E-mail: _____

Информация для выставления счета организациям

Название _____ ИНН _____
Телефон/факс (_____) _____
Контактное лицо _____
Адрес для отправки заказа: индекс _____ страна _____
область _____ город, поселок _____
улица _____ дом _____
адрес электронной почты (если он у Вас есть) : E-mail: _____



БЛАНК ЗАКАЗА

(принимаются ксерокопии)

Автор.....	Название.....	Цена Россия (руб.)	Цена Украина (грн.)	Год.....	Объем	Заказ (экз.)
Серия: Компьютерная шпаргалка						
Егоров.....	МиниЖелтые страницы Интернет. Компьютерная шпаргалка ..	18	5	2006	80	
Егоров.....	Поиск в Интернет. Компьютерная шпаргалка.....	18	5	2006	80	
Золотарева	Электронная почта. Компьютерная шпаргалка	18	5	2006	80	
Колосков	Microsoft Windows XP. Компьютерная шпаргалка.....	18	5	2006	80	
Кузнецова	Microsoft Word 2003: работаем с текстом.....	18	5	2006	80	
Матвеев	Вычисления и расчеты в Excel 2003. Комп. шпаргалка	18	5	2006	80	
Юдин.....	Microsoft Excel 2003: работаем с таблицами	18	5	2006	80	
Серия: Просто о сложном						
Алешков	Программы-переводчики. Осваиваем сами	54	12	2005	144	
Антоненко	«Толстый» самоучитель работы на компьютере	131	29	2005	544	
Воробьев.....	Nero Burning ROM. Записываем CD и DVD	54	12	2005	192	
Главенка	45 лучших программ для созд., копир., обработки CD и DVD ...	143	32	2005	368	
Дмитриев	Настройки BIOS. Изд. 2-е	76	17	2005	288	
Жарков.....	AutoCAD 2004. Эффективный самоучитель. Изд. 2-е	164	37	2005	560	
Жарков.....	AutoCAD 2005: Эффективный самоучитель.....	173	39	2005	600	
Золотарева	Желтые страницы Интернет 2006: Лучшие русские ресурсы ..	94	20	2006	368	
Колисниченко.....	Самоучитель PHP 5. 3-е издание	182	40	2005	576	
Колисниченко.....	Самоучитель LINUX. Установка, настр., использ. Изд. 4-е.	175	39	2006	688	
Колисниченко.....	Сделай сам комп. сеть. Монтаж, настройка, обслуж. Изд. 2-е ...	142	29	2004	448	
Колисниченко.....	Англо-русский толковый словарь компьютерных терминов.....	83	18	2006	272	
Колосков	Windows XP. Популярный самоучитель. Изд. 2-е. перер. и доп..	108	24	2005	368	
Кузнецова	Установка и переустановка Windows. Изд. 4-е.....	43	10	2005	128	
Кузнецова	Microsoft Windows XP. Краткое руководство	63	14	2005	256	
Лохниски.....	222 проблемы работы на компьютере и их решение.....	65	39	2006	224	
Марек.....	Ассемблер на примерах. Базовый курс	109	24	2005	240	
Матвеев	Самоучитель MS Windows XP. Все об использ. и настр. Изд. 2-е ..	160	36	2006	624	
Пестриков	Турбо Паскаль 7.0. Изучаем на примерах. Изд. 2-е	109	22	2004	368	
Подольский	Печать на ПК слепым десятипальцевым методом. Изд. 3-е	28	6	2006	96	
Пономарев	Самоучитель работы на компьютере + цв.вклейки	106	24	2005	368	
Серогодский	Excel 2003 + цв.вклейки . Эффективный самоучитель. Изд. 2-е ..	153	34	2006	400	
Сухарев.....	Turbo Pascal 7.0. Теория и практика программир. Изд. 2-е.....	165	34	2004	640	
Юдин.....	Самоучитель работы на ноутбуке. Изд. 2-е + цв.вклейки	175	39	2006	512	
.....	Скачиваем фильмы, музыку и программы из Интернета.					
.....	Пиринговые сети.....	65	15	2006	272	
.....	Реестр Windows XP: Трюки и секреты.....	65	15	2006	192	
Серия: Полное руководство						
Досталек	TCP/IP и DNS в теории и на практике. Полное руководство.....	263	58	2006	608	
Колисниченко.....	Linux: Полное руководство	252	54	2006	784	
Шетка.....	Microsoft Windows Server 2003: Полное руководство.....	241	53	2006	608	
Серия: Секреты мастерства						
Колисниченко.....	Linux-сервер своими руками. Изд. 4-е перер. и доп.....	219	45	2005	752	
Мозговой	Классика программирования: алгоритмы, языки, автоматы, компиляторы. Практический подход	175	37	2006	320	
Сухарев.....	Основы Delphi. Профессиональный подход.....	184	37	2004	600	
Финков.....	Интернет. Шаг второй: от пользует. к профессионалу + CD ..	88	17	2002	768	
Юдин.....	Ноутбук: особенности испол. и настройки + цв.вклейки	182	41	2005	384	



(принимаются ксерокопии)

Автор.....	Название.....	Цена Россия (руб.)	Цена Украина (грн.)	Год.....	Объем	Заказ (экз.)
Серии: Профи и др.						
Вебер.....	Knowledge-технологии в консалтинге и управл.предпр.+ CD ..	127	18	2003.....	176	
Гургенидзе	Мультисервисные сети и услуги широкополосного доступа	87	18	2003.....	400	
Есипов	Информатика (учебник). Изд. 3-е	102	21	2003.....	400	
Куприянов	Техническое обеспечение цифровой обработки сигналов.....	66	12	2000.....	752	
Кучеров	Источники питания ПК и периферии. Изд. 3-е	142	32	2005.....	432	
Кучерявый	Пакетная сеть связи общего пользования	109	27	2004.....	272	
Кучерявый	Управл. трафиком и качество обслуживания сети Интернет	128	29	2004.....	336	
Никамин	Цифровая звукозапись. Технологии и стандарты	нет	12	2002.....	256	
Щеглов.....	Защита комп. информации от несанкционир. доступа	164	20	2004.....	384	
Серия: Библиотека пользователя						
Будилов.....	Работаем с FINALE 2001 + CD	50	10	2001.....	240	
Серогодский	Просто о сложном. Windows 98.....	33	6	2001.....	144	
Серогодский	Просто о сложном. Word 2000	нет	6	2001.....	224	
Серия: Конспект программиста						
Будилов.....	Практические занятия по PHP 4 + CD	33	9	2001.....	352	
Костельцев	Построение интерпретаторов и компиляторов + CD	44	9	2001.....	224	
Куправа	Самоучитель Access 97/2000 + диск	43	8	2001.....	144	
Николенко	MIDI — язык богов + CD	39	8	2000.....	144	
Серия: Домашний мастер						
Балясников	Обувь. Выбор, уход, ремонт	32	9	2003.....	240	
Бадьин	Справочник Строителя-технолога	132	30	2005.....	320	
Бирюков.....	Защита автомобиля от угона	61	10	2003.....	176	
Богданов	Умный дом. Изд. 2-е.....	87	19	2005.....	208	
Давиденко	Настольная книга дом. электрика. Люмин. лампы.	98	22	2005.....	224	
Дворецкий	Автомобильные сигнализации	175	38	2006.....	544	
Корякин-Черняк	Холодильники от А до Я. Изд. 2-е, перераб. и дополн.	123	20	2003.....	416	
Корякин-Черняк	Справочник домашнего электрика. Изд. 4-е, перер. и дополн.	142	29	2006.....	400	
Корякин-Черняк	Освещение квартиры и дома	76	17	2005.....	192	
Корякин-Черняк	Краткий справочник домашнего электрика	87	19	2005.....	272	
Корякин-Черняк	Современные автосигнализации	158	35	2006.....	400	
Ландик	Современные холодильники NORD	72	10	2003.....	144	
Пестриков	Новейшая азбука сотового телефона. Изд. 3-е.....	142	32	2005.....	352	
Пестриков	Домашний электрик и не только. Кн. 1. Изд. 4-е	87	19	2005.....	224	
Пестриков	Домашний электрик и не только. Кн. 2. Изд. 4-е	87	19	2005.....	224	
Торопкин	Ламповый Hi-Fi усилитель своими руками.....	120	27	2005.....	240	
Серия: Радиоловитель						
Белов	Конструирование устройств на микроконтроллерах.....	98	20	2005.....	256	
Виноградов	Микропроцессорное управление телевизорами	55	11	2003.....	144	
Петров	Звуковая схемотехника для радиоловителей.....	109	16	2003.....	400	
Саулов	Телевизоры: ремонт, адаптация, модернизация. Изд. 2-е.....	125	28	2005.....	320	
Саулов	Металлоискатели для любителей и профессионалов	87	18	2004.....	224	
Семьян.....	500 схем для радиолубит. Приемники. Изд. 2-е перер. и доп.	87	19	2005.....	272	
Семьян.....	500 схем для радиоловителей. Источники питания. Изд. 2-е	130	25	2006.....	416	
Шмырев	Радиостанция своими руками + вклейка	76	12	2004.....	144	



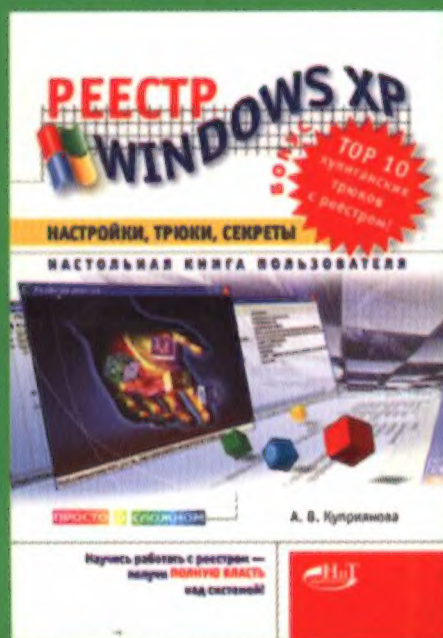
(принимаются ксерокопии)

Автор.....	Название.....	Цена		Год.....	Объем	Заказ (экз.)
		Россия (руб.)	Украина (грн.)			
Серии: Телемастер и Энциклопедия телемастера						
Безверхний	Телевизоры DAEWOO и SAMSUNG + схемы	153	17	2003	144	
Виноградов	Импульсные источники питания видеоманитонов	65	9	2003	160	
Корякин-Черняк	Применение телевизионных микросхем. Т. 1	164	29	2004	320	
Корякин-Черняк	Применение телевизионных микросхем. Т. 2	164	29	2004	304	
Корякин-Черняк	Применение телевизионных микросхем. Т. 3	164	37	2005	320	
Пьянов	Тел. LG на шасси MC-51B, MC-74A, MC-991A + схемы	131	13	2003	144	
Пьянов	Видеопроцессоры семейства UOC + схемы	109	13	2003	160	
Рязанов	1001 секрет телемастера. Кн. 1. Изд. 2-е, перер. и доп.	142	32	2005	304	
Рязанов	1001 секрет телемастера. Кн. 2. Изд. 2-е, перер. и доп.	142	32	2005	224	
Саулов	Переносные телевизоры	142	12	2002	512	
Янковский	Импульсные источники питания ТВ. Изд. 2-е перер. и доп.	142	32	2005	280	
Серия: Электронные компоненты						
Герасимов	Интегральные усилит. низкой частоты. Изд. 2-е, перер. и доп. ...	175	13	2003	528	
Партала	Цифровые КМОП-микросхемы. Справочник	83	13	2001	400	
.....	Микросхемы для CD-проигрывателей. Сервосистемы	109	18	2003	272	
.....	Видеопроцессоры. Справочник	142	18	2004	256	
.....	Зарубежные м/сх, транзисторы, тиристоры, диоды + SMD. 0...9.					
.....	Справочник, Изд. 3-е перераб. и доп.	241	54	2005	672	
.....	Зарубежные м/сх, транзисторы, тиристоры, диоды + SMD. A...Z.					
.....	Том 1 (A...M). Справочник, Изд. 3-е перераб. и доп.	241	54	2005	656	
.....	Зарубежные м/сх, транзисторы, тиристоры, диоды + SMD. A...Z.					
.....	Том 2 (N...Z). Справочник, Изд. 3-е перераб. и доп.	241	54	2005	688	
.....	Мощные транзисторы для телевизоров и мониторов.	197	44	2005	448	
.....	ТВ микросхемы. Т.1: ИМС обработки ТВ сигналов	142	22	2004	288	
.....	ТВ микросхемы. Т.2: ИМС источников питания	142	32	2005	288	
.....	ТВ микросхемы. Т.3: ИМС обработки сигн. звук. сопровожд. ...	142	32	2005	240	
.....	ТВ микросхемы. Т.4: ИМС для систем разверток	142	32	2005	208	
Серии: Радиомастер и др.						
Азраменко	Ремонт и регулировка CD-проигрывателей + вклейки : 13 схем	58	нет	1999	160	
Брускин	Зарубежные резидентные радиотелефоны. Изд. 2-е	нет	7	2000	176	
Брускин	Схемотехника автоответчиков + вклейки : 14 схем	нет	7	1999	176	
Виноградов	Источники питания видеоманитонов	50	10	2001	256	
Виноградов	Источники питания видеоманитонов и видеоплейеров	нет	6	1999	128	
Виноградов	Сервисные режимы телевизоров : 1—12, цена за 1 том	55	10	2002	208	
Журавлев	Тел. процессоры системы управления. Изд. 2-е, дополн.	83	13	2001	512	
Заикин	Современные радиотелефоны (ТА от А до Я. Кн. 4)	175	20	2004	352	
Корякин-Черняк	Телеф. аппараты от А до Я. Кн. 1: Абон. телеф. аппараты., Изд. 5-е	121	18	2003	368	
Корякин-Черняк	АОН в телефонных аппаратах, Изд. 2-е (ТА от А до Я. Кн. 2)	109	18	2003	336	
Котенко	Электрон. телефон. аппараты, Изд. 3-е (ТА от А до Я. Кн. 3)	106	17	2003	272	
Котенко	Электронные телефонные аппараты. Изд. 2-е, перераб.	нет	9	2001	192	
Кульский	КВ-приемник мирового уровня	40	6	2000	352	
Кучеров	Источники питания мониторов	нет	9	2001	240	
Лукин	Источники питания моноблоков и телевизоров	нет	6	1998	136	
Лукин	Телевизоры ближнего зарубежья	нет	6	1998	136	
Мухин	Энциклопедия мобильной связи	63	7	2001	240	
Партала	Видеокамеры + вклейки : 12 схем	нет	7	2000	192	
Пестриков	Энциклопедия радиолюбителя. Работаем с ПК	87	18	2004	208	
Пестриков	Радиоэлектроника в конструкциях и увлечениях	87	18	2004	256	
Янковский	Видеомагнит. серии ВМ. изд. 2 + вклейки : 5 схем АЗ	66	11	2000	272	
Цены розничные, указаны без учета почтовых расходов. Цены действительны до 31 декабря 2006 года.						



В а ш п р а в и л ь н ы й в ы б о р !

Книги серии
"ПРОСТО О СЛОЖНОМ"
от издательства
"НАУКА И ТЕХНИКА"



Вольский С. В., Дмитриев П. А

TURBO PASCAL 7.0

для студентов и школьников

Эта книга представляет собой отличное учебное пособие по программированию на языке Turbo Pascal 7.0. Предназначена прежде всего для студентов и школьников, так как содержит весь необходимый объем информации, в то же время не перегружая излишними деталями. Удобная структура и наличие шпаргалок-напоминалок по каждой теме делают книгу незаменимой как при самостоятельном изучении Turbo Pascal, так и при использовании в учебном процессе.

Книга написана доступным языком опытными преподавателями. Содержит большое количество примеров с подробно откомментированными строками программы. Лучший выбор для эффективного изучения Turbo Pascal и подготовки к различным зачетам, контрольным работам и т.п.

Россия: Санкт-Петербург
для писем: 192029, Сан
(812) 567 7025, 567 702
e-mail: nit@mail.wplus.n

Украина: 02166, Киев, у
(044) 516 3866
e-mail: nit@volicable.com

